1 INTRODUCTION

In embedded system design a trend towards high integration of different functionality on the same platform can be observed across a wide-range of industries as e. g. automotive, aerospace, and industrial automation. This trend is driven by cost pressure and increasingly shorter product cycles, and it is made possible through a number of technological advances.

Multi-core architectures, which are known from general purpose computing for some years, have found their way into industrial practice of embedded system design. They provide greatly increased compute power at moderate energy consumption. This allows to consolidate functionality that is traditionally distributed across several electronic control units (ECUs) onto a common processing element, thus giving op-

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

portunities to significantly reduce hardware, wiring and energy cost. Similarly, the availability of low-cost high-speed communication networks that are suitable for use in embedded systems allows to reduce the number of networks in today's complex systems. E. g. the FlexRay bus [41] provides high-speed real-time capable communication for automotive networks. For the future, widespread use of switched Ethernet as backbone bus in automotive networks is expected to further reduce the number of buses [141]. In the domains of aerospace and industrial automation, real-time capable Ethernet derivatives are already being used today with e. g. the ARINC 664 [1, 139] and the ProfiNet [116] standards. These technological advances provide the required performance to integrate more functionality on a platform.

The increased system complexity that arises through a higher degree of integration has to be addressed in the design process – especially in the face of shortened product cycles. One key step to tackle this complexity is to efficiently allow for design reuse, i. e. use of previously developed components in a new system. A second measure is to ease distributed development to parallelize workflows. Both measures can be achieved by defining clear-cut interfaces among components. The third key step is to provide easy-to-use mechanisms for composing these individual components based on their interfaces. This can be achieved e. g. through automated property verification or system synthesis steps.

The integration problem outlined above, however, is more complex than a mere composition of application programming interfaces (APIs) of components. In embedded systems application requirements go beyond functional requirements due to the close interaction with the physical environment. Systems have tight constraints on their real-time behavior as e. g. printing machines in industrial automation, where the single motors of the paper feed have to be accurately synchronized to avoid paper jams. Such real-time constraints can come along with requirements on system safety, as e. g. for the brakes in automotive break-by-wire applications, or with requirements on availability as e. g. in the flight control of an aerospace system. These constraints, as well as the component properties that determine their adherence, have to be modelled in the component interfaces. The composition of a complex embedded system then requires to determine system-level properties from the component interfaces, which have to incorporate non-functional properties and constraints. In many cases, as e. g. real-time, the system-level properties resulting from component composition are super-additive and often not even continuous, i. e. a system-level property cannot be determined as sum of component properties and an infinitesimal value change of a component specification can have a large impact on a system property. Thus, the adherence of components to their interface is critical when system-level properties have to be guaranteed.

In the face of this problem additional issues arise with a higher degree of integration. Consolidating several components on the same platform introduces additional dependencies through use of shared resources. In safety-critical systems, however, certification may only be performed on a sub-system level if "sufficient independence"

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

between components is guaranteed [62]. Otherwise, all components must be certified according to the highest applicable safety standard. This can be an extremely costly design process. Thus, when two components of different safety-criticality shall be consolidated on a common platform, they either must be certified jointly or appropriate mechanisms that guard component behavior must be used, i. e. the component specification/interface has to be enforced. Such mechanisms must be simple as they become subject to certification themselves.

A second issue arises, when the interface of a component that shall be integrated inherently cannot be characterized appropriately. This may be the case when e. g. the component displays behavior, which is adaptive to its application context. Typical examples for this are e. g. components that highly depend on user-interaction or variable bitrate video de-/compression algorithms [37], whose computational complexity depends on the considered scene. Component interfaces also can be inherently hard to characterize when open networks (e. g. Car-to-X communication) are considered, or when components can be updated or exchanged in a plugin-style manner. Although the system integrator may specify interfaces for such components, the conformance of the components to the interfaces cannot be validated at design-time.

Thus, for coming highly integrated embedded systems we expect to see

- a larger number of components accessing the same shared resources,
- components with different qualification or certification requirements, and thus with different confidence in their specification,
- and components that inherently cannot be specified accurately.

Nonetheless, the component specification retains its importance for verification of system-level properties due to their super-additive and non-continuous nature. The need for accurate specification is indeed even aggravated through the high number of components. This thesis addresses this ambivalence in the scope of safety-critical real-time systems. Particularly, we investigate how the conformance of components to their specification can be enforced efficiently through runtime mechanisms. In this scope, we will follow an approach of runtime monitoring.

In the remainder of this chapter we elaborate further on the context of this thesis. Particularly, we discuss the state-of-the-art in embedded system design with a particular focus on safety and real-time properties and the aspects of component reuse and integration. Further, we address the issue of incorporating components with inaccurate and changing specification – two aspects that have gained particular relevance lately. Based on this discussion of current developments in the design and architecture of embedded systems, we define the research objective of this thesis.

1.1 DESIGN OF EMBEDDED SYSTEMS

To manage the design complexity of today's embedded systems, proper engineering methods are fundamental. In the automotive industry, which we regard representatively

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

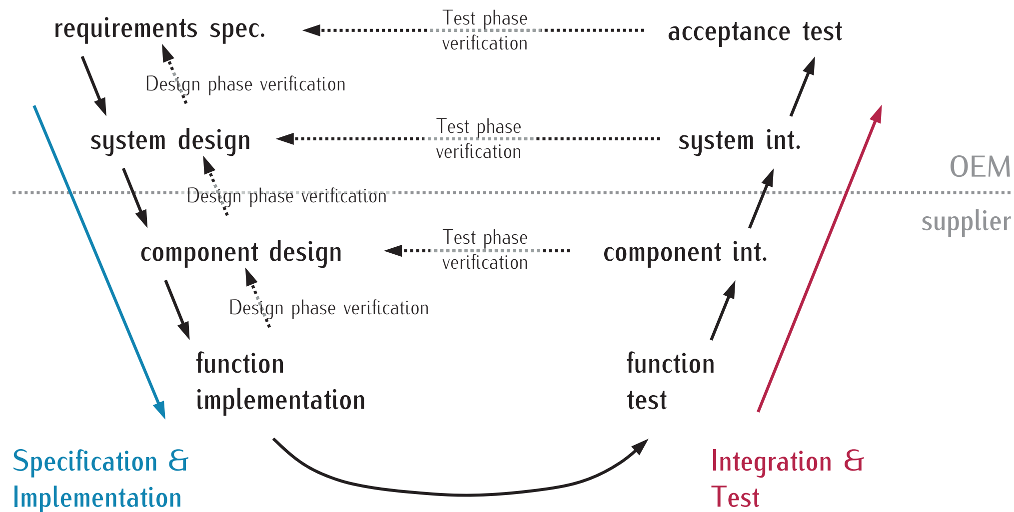

Figure 1.1: Design Process according to the V-Model

for the domain of safety-critical systems, development follows the "V-Model" [60, 63, 125] as shown in Figure 1.1. This design process follows a top-down approach, in which, starting from an initial requirements specification, the system design is step-wise broken down and refined into components and functions of manageable complexity. The components and functions can then be implemented concurrently by different teams or suppliers according to their specification. This implementation branch is mirrored in a corresponding integration & test branch to guide the composition of the independently developed components and functions. Each function, component, system and finally the product is tested for satisfaction of its specification. Comparable design processes, following at least the specification and implementation branch of the V-Model can be found in other standards as the generic safety standard IEC 61508 [62] or the safety standard for airborne systems DO178-B [121].

In the requirements specification phase of the V-Model high level application requirements are defined per intended system function. In the special case of safety-critical applications this step also already incorporates safety requirements. Specifically, the automotive standard for functional safety ISO 26262 [63] requires identification of functions that could lead catastrophic failures, e. g. "functions that enable the system to achieve or maintain a safe system state", "functions related to detection [. . .] of faults" or "functions that allow modifications of the software".

In the system design phase ("software architectural design" in ISO 26262-6 [63], "software design process" in DO178-B [121]) the requirements are mapped to architectural entities. This includes static aspects, such as software structure and datatypes, as well as dynamic aspects of the design, such as control flow and concurrency of processes or temporal constraints. In the component design phase [125] the design is further subdivided in a hierarchical manner.

Finally, in the function implementation phase the specified components/functions are implemented according to the specification. This can be done either directly in

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

source code or in a modelling language with appropriate code generation. The product of this stage is source code and a translation to object code.

In the corresponding integration and test branch of the V-model the implementation artifacts are tested for conformance with their respective specification and are integrated into a composed system. On the function, component and integration test levels, this incorporates function, interface, and resource usage tests as well as fault injection tests for safety mechanisms, for the corresponding stage of integration. On the acceptance test level the complete composed system is tested w.r.t its specification. This includes hardware-in-the-loop (HIL) tests, restbus simulations as well as tests of the final product.

This "V-model" process bears the caveats that

1. inconsistencies and errors in the initial specification are only detected relatively late in the design process

2. the requirements specification has to be available at the beginning of the design.

To alleviate the first issue and to detect inconsistencies in the specification early on, the specification and implementation branch is extended by design phase verification steps [63] as shown in Figure 1.1 [1]. Their purpose is to show that the further specification/implementation of a function is indeed a refinement of the higher level design. These design phase verification steps can be performed e. g. through formal verification, data flow and control flow analysis, simulation (of an executable model) or prototype generation [63]. The choice of the employed methods depends on the design phase, the safety criticality of the considered component or function and the used modelling/implementation language.

The second issue has been addressed in an updated version of the V-Model called "V-Model XT" [61], which allows tailoring of the development process to specific needs. This includes e. g. iterative refinement of the system specification.

At all stages of the design process also non-functional requirements, such as real-time and safety properties as considered in this thesis, must be taken into consideration and have to be specified if applicable.

1.1.1 *Functional Safety*

In the application domains of automotive and aerospace systems stringent safety standards apply. The applicable standards, i. e. IEC 61508 [62] as application-independent standard, ISO 26262 [63] for the automotive, and DO178-B [121] for the aerospace domain, not only stipulate requirements and guidelines for the developed system but also on its design process.

1 sometimes the design phase verification steps are represented as a second "V" of a *virtual integration* [130]

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

The requirements on the design process address every stage of the "V-model". For the requirements definition phase e.g. IEC 61508 [62] suggests use of tools, such as *Controlled Requirements Expression (CORE)* [90], to ensure that all design requirements are properly captured and consistent. Further, it suggests use of certain modelling schemes and languages (e.g. data flow diagrams, Higher Order Logic (HOL), temporal logic) and development and coding practices (e.g. modular software design, wrappers for library functions, avoidance of dynamic variables and objects, limited use of interrupts, pointers and recursion). Some of the required design practices can become extremely cost intensive e.g. the use of *multiple-version dissimilar software* [121] (or *independent parallel redundancy* in [63]) where several independently developed versions of the same functionality are required to provide additional safety [121]. Additionally, for certain safety levels the standards request that implementation, and verification and test be performed by independent teams (e.g. *formal inspection* in [62]).

All these design process requirements make the development of safety-critical systems extremely costly. As safety certification requires consideration of the complete integrated system rather than individual components, the certification effort and cost for highly integrated systems rises dramatically. This is aggravated when several functions that shall be integrated on a common platform – as envisioned for modern multi-core processors – have different safety criticalities. In this case all components must be developed according to the highest applicable standard if no *sufficient independence* [62] (or *freedom from interference* [63]) can be guaranteed. In this context sufficient independence refers to the guarantee that a fault of a component with low safety-criticality cannot cascade down to a component of higher criticality. Thus, the standards require certification of a non-safety-critical component if it cannot be guaranteed that a failure cannot affect another safety-critical component.

In order to reduce certification effort, all of the mentioned standards suggest methods to establish such sufficient independence among components through architectural means. One such architectural measure is *partitioning* [121, 63], where additional hardware or software components prevent interaction among partitioned components. In this case the partitioning mechanism must be certified according to the highest applicable safety level. Such partitioning has to consider hardware resources (e.g. processors, memory devices, interrupts), control and data coupling, and failure modes of components and protection mechanisms. Techniques, that are typically used for such partitioning, include e.g. spatial isolation through memory protection units (MPUs) or memory management units (MMUs) [69] and temporal isolation through time division multiple access (TDMA) scheduling [115] or methods as employed in the *Time-Triggered Architecture* [70]. Partitioning is also applied to communication networks as e.g. in the scheduling of *FlexRay* [41] or *Time-Triggered Ethernet* [71]. Strict partitioning pre-allocates system resources per partition and does not allow reclamation of unused resources. As a consequence it causes considerable overhead.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

On the other hand it makes certification independent of the specification of an isolated component, and thus eases verification.

Safety Monitoring [121] is another measure to guarantee sufficient independence. A dedicated monitor, which can be implemented in hardware, software or a combination of both, directly monitors a function for failures. Among others, such a monitoring may regard functional behavior (e. g. sanity check of output values), access to resources (e. g. memory access patterns) or temporal properties (e. g. maximum activation frequency). Monitors may be used to check a component directly w.r.t. a given specification, or in a design that follows the principle of multiple-version dissimilar software, w.r.t the outputs of two or more distinct implementations of the same component. The technique of monitoring can be used to reduce the safety requirements of the monitored function, while the monitor must be certified to apply this technique. This requires verification of the correctness of the monitoring mechanism, an assessment of the fault coverage, and a proof of independence of monitor and function. Monitoring does not pre-allocate system resources to single components, as strict partitioning does. Thus, it potentially allows for higher resource utilization – particularly if it checks against a given specification, rather than the outputs of multiple instances of a component.

The specification of properties at the external interfaces of a component, which is required for monitoring, is also required in the specification and design verification steps of the V-Model (cf. Figure 1.1). Further, such specifications play an important role for the reuse of component as discussed in the following section.

1.1.2 *Design reuse*

Due to cost and time-to-market pressure, high design efficiency is important. To achieve this goal the reuse of previously designed components is key. Development according to the V-Model is tailored to allow design reuse through its hierarchical design procedure and the refinement of components. This is widely exploited in industry.

In the automotive domain, original equipment manufacturers (OEMs), i. e. car manufacturers, defer the responsibility for entire subsystems or single components to suppliers, which develop components according to specification [125], i. e. the responsibilities in the V-Model are separated by horizontal cuts (Figure 1.1). If the specification permits, these components or subsystems are integrated into different vehicles – also from different vendors. As a result such components can be developed more cost efficiently. Also vice-versa, the integrator typically has different suppliers for the same component (i. e. identical specification), which allows to reduce cost and to reduce dependency on single suppliers. Thus, as long as components or subsystems fulfill their external specification – or *interface* as we will later call it – they can be readily reused.

For safety aspects, design reuse raises additional issues, though. Certification requires to regard a system as a whole. As a consequence isolated components

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

cannot be certified. However, e. g. ISO 26262 [63] specifically addresses the issue of component reuse. Components that are reused with modifications require to be certified along the standard process. However, components that are reused without modifications and are sufficiently qualified can be reused in a new system. In this case, the specification of a qualified component requires to include functional requirements, behavior in case of failures or overload, response time of the component and information on the resource usage and on the runtime environment (RTE). Further, the interfaces and the description of the component integration and configuration have to be included.

Thus, also the reuse of components is a specification-driven process. This specification requires to include all relevant information for the execution context of the respective component. As a result, the assumptions that were made for the design phase verification steps and the test phase verification steps still hold for the reused component and for the remainder of the system.

1.1.3 *Functional Architectures and Consolidation of Communication*

In order to further aid in the integration of complex embedded systems and to foster the reuse of legacy components, the use of integration platforms is promoted across several industries. This includes *functional architectures* on the software side as well as *consolidation of networks* on the communication side. In either case the integration platform is closely coupled with the modelling and verification of the above design processes.

In the automotive industry this trend can be observed with the standardization of a common operating system interface and middleware. In 1993 the Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (OSEK) consortium, consisting of several German automotive companies, started a standardization initiative for distributed control units in automotive systems. In 1994 the French Vehicle Distributed eXecutive (VDX) consortium joined the effort. One of the main results is a common operating system interface OSEK-OS [49] for automotive systems. The standard provides a specification of an application interface for operating system and input/output (I/O) services. The main purpose of this standardization being better portability of software components among platforms. With the provided interface specification software components are being made largely independent of the underlying hardware platform and base-software. Further, to increase portability the OSEK/VDX consortium defined a language for specification of configuration options with the OSEK Implementation Language (OIL), which can be automatically processed and compiled into code by appropriate design tools [160]. In the light of the above design process, we see that these standardization efforts attempt to harmonize component interfaces in order to ease component reuse according to currently used safety standards [63].

The efforts for alleviating integration effort in the automotive domain are continued in the AUTomotive Open System ARchitecture (AUTOSAR) consortium [11]. Based on the OSEK standards the specification of an automotive middleware interface with an

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

associated design process are developed [12]. It allows the synthesis of a RTE based on a specification of software components and their communication dependencies. Further, it specifies some mechanisms for monitoring components according to their specification.

Comparable approaches for integration can be found in the aerospace industry. The aerospace industry standardized a common real-time operating system (RTOS) interface in 1997 in the ARINC 653 [7] standard [115]. The Application/Executive Interface (APEX) of ARINC 653 defines the way in which software components interact with the RTOS, again providing a well-defined interface among components to ease the integration process. In contrast to the automotive domain the degree of integration has been driven further, due to the requirement to reduce the number of hardware components to reduce weight and energy consumption [115]. Integrated Modular Avionics (IMA) architectures aim at integrating different functionality of an aircraft on common hardware, i. e. to introduce the extensive use of shared resources [43]. As the integrated components may have different safety certification requirements according to the applicable safety standard DO178-B [121], ARINC 653 readily provides means for component isolation through strict segregation. This is established via virtual memory in the spatial domain and via TDMA scheduling in the temporal domain [7]. Thus, also here some means that can provide *sufficient independence* are part of the operating system specification. As a consequence, the operating system, which represents a pre-qualified component, can be reused across several projects, and it isolates components to reduce the certification effort for the components. In other words, the verification requirements of the integration & test branch of the V-Model are minimized by architectural means, which can be used as pre-qualified mechanisms that do not require re-certification.

The trend towards high integration is not limited to software systems, but can also be observed for communication media. In the scope of the IMA the aerospace industry defined Avionics Full Duplex Switched Ethernet (AFDX) communication network as part of the ARINC 664 standard [1]. One of its aims is to reduce the number of buses in an aircraft by consolidating the communication on fewer media. A key concept to maintain the high safety qualification of the legacy bus configurations is the definition of Virtual Links (VLs), which are isolated against each other and can be associated with certain quality-of-service (QoS) guarantees. The previously used legacy buses are routed transparently through the AFDX network. Although the transparent use for legacy devices eases integration to some degree, it is still a painstaking process. The configuration of an AFDX network requires careful configuration of the QoS guarantees for each VL and even requires separate configurations for each router in the network.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

1.2 EVOLVING SYSTEMS AND INACCURATE SPECIFICATION

Above we have seen that the design process for critical embedded systems is governed by specifications, modelling and description of interfaces. The consistency of the specification and the integration of components to a complete system is verified thoroughly at design time. Through this process the adherence of all requirements is ensured.

As a result from this methodology, safety-critical systems are configured statically and provide little room for adaptation after deployment. DO178-B [121] provides detailed guidelines for modifications of previously developed software (comparable to the guidelines on component reuse in ISO 26262 [63]), stating that the safety assessment has to be re-evaluated considering the modifications. Further, for user-modifiable software it states, that it must be shown that any modification through the user cannot affect system safety – independent of the correctness of the user-modified software. Additionally, upon user-modification of a component, the user assumes all responsibility of the safety of the modified component. Comparable requirements apply to *option-selectable components* and *field-loadable software*. For the latter, mechanisms to ensure detection of corrupt or partial loading of software and compatibility of the software are required.

As a consequence from these requirements on software adaptation, deployed systems are rarely changed [133]. When they are, the changes usually affect non-critical components that are guarded by means to establish sufficient independence. This way the adherence to certain parts of the specification of a component is ensured by protection mechanisms (e. g. partitioning or monitoring) to compensate for the unknown specification of future system updates.

Comparable problems arise when components cannot be accurately specified right from the beginning of the design process. This is the case when e. g. the component displays behavior that is adaptive to its application context. Examples for this are components that highly depend on user-interaction (i. e. user interface) or variable bitrate video de-/compression algorithms [37], whose computational complexity depends on the considered scene. A component specification can also be inherently hard to characterize, when open networks (e. g. Car-to-X communication [160]) are considered. Although the system integrator may specify interfaces for such components and enforce the behavior through appropriate runtime mechanisms, the definition of suitable interfaces is hard. This is especially true when a component's typical behavior significantly deviates from its worst-case behavior (e. g. best-effort applications). Allocating resources according to the typical component resource requirements may cause a deterioration of the component's perceived performance, while an allocation to the worst-case requirements may cause severe overprovisioning in the typical case. For either case strict partitioning mechanisms, which statically pre-allocate resources, are sub-optimal.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.