Towards Verified Automotive Software*

J. Botaschanjan, L. Kof, C. Kühnel, M. Spichkova
Institut für Informatik
Bolzmannstr. 3
D-85748 Garching, Germany
{botascha, kof, kuehnelc, spichkov}@in.tum.de

ABSTRACT

Automotive software is one of the most challenging fields of software engineering: it must meet real time requirements, is safety critical and distributed over multiple processors. With the increasing complexity of automotive software, as for example in the case of drive-by-wire, automated driving and driver assitents, software correctness becomes more and more a crucial issue. In order that these innovations can become reality, it is necessary to be able to *guarantee* software correctness.

The presented work aims at verification of automotive software. For this purpose it introduces a verification approach, including a framework of verified modules which assists the verification of the actual application. Feasibility of this approach was validated on a case study that also showed how verification can be integrated into the development process.

Categories and Subject Descriptors

C.3 [**Special-Purpose And Application-Based Systems**]: Real time and embedded systems; D.2.4 [**Software Engineering**]: Software/Program Verification—*Correctness proofs, Formal methods*

General Terms

Verification, Design

Keywords

Automotive, Integration, Verification, Model Based Software Engineering, Time-Triggered, Theorem Proving

*This work was fully funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author(s).

1. INTRODUCTION

Over the last years the trend to embed more electronics and software functionality into the vehicles considerably advanced the complexity of the software applications for automotive control. Along with that an increased demand for improving the reliability of this functionality comes. Facing these challenges new approaches in ensuring the correctness of the software systems have to be considered.

Safety critical real time applications distributed over a network of embedded systems are typical for this domain. Thus a safety critical function is typically provided by a set of subsystems deployed on a heterogeneous network of electronic control units (ECUs). These are often developed in parallel by different suppliers. In the integration phase of the development it has to be ensured that the combination of the implemented subsystems realises the desired overall behavior. For this purpose *testing* is the state of the art technique in the automotive industry. However, testing can only exemplarily demonstrate the absence of errors, not the correctness. Due to the high demand for the reliability of the safety critical systems this is insufficient. In opposite to testing, *verification* delivers a correctness proof for the fulfillment of safety critical properties by the system.

To come up with this proof, development has to get on with a number of challenges. At first the affected subset of functionality must be identified and isolated. Secondly, the correctness of the demanded functionality has to be proven. Finally, it must be guaranteed, that the rest of the system cannot interfere with this safety critical subset.

Every identified subsystem is embedded into two kinds of environments: its communication partners (local or remote) and its run time environment, consisting of the operating system, the communication mechanisms and the hardware the task runs on[1]. This implies that the correctness of a safety critical system depends on the correctness of every involved subsystem, the correctness of their *integration*, as well as on the correctness of their execution and communication infrastructures.

In the described problem space the goal of the Verisoft Automotive project [17] is to ensure the quality of this kind of systems by adding formal verification to their development process. The proposed model based approach aims to guarantee the correctness of the developed systems and to reduce the verification and integration efforts at the same time.

[1]Also in the case that the hardware is transparent to the applications, it is still the main factor determining their execution time and other resource usages.

The approach provides a *verification framework* with a formalized verified operating system (OSEKtime) and a communication protocol (FlexRay) (see Section 3). Further on the framework simulates the behavior of the hardware portion of the developed system (e.g. of the ECU and network). Using this framework, models of developed applications can be verified for the fulfillment of desired properties. The framework verifying single subsystems, their embedding into the run-time environment, as well as the integration of several subsystems. At the same time C code can be generated from the same models, which can executed in the environment of this OS/bus combination and accomplish its mission correctly. The approach was evaluated with a case study. Within its scope the *emergency call* system was modeled and verified.

The following section of the paper introduces the approach and discussed its integration into development process. Section 3 presents the verification framework and Section 4 describes its application within the emergency call case study. Section 5 overviews the related work. Finally Sections 6 and 7 summarize the presented work and sketch further research directions.

2. DEVELOPMENT METHODOLOGY

The development methodology proposed here consists of 4 phases: natural language requirements in the specification phase (Section 2.1), a model in the design phase (Section 2.2), theories in the verification phase (Section 2.3) and C code in the integration phase (Section 2.4). In the following sections each of these phases will be described along with the workflow products for each step. The phases are illustrated in Figure 1.

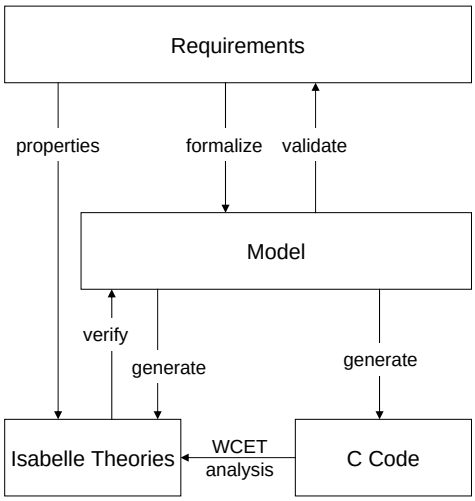

Figure 1: Work Products

2.1 Specification

The input of the presented approach is a natural language specification. The *requirements* describe the desired behavior of the application to be developed. In addition to these requirements, the specification contains a set of *properties*, which deliberately specify safety critical aspects of the system. The properties of the software to be developed will be proved by formal methods in later stages.

2.2 Design

The requirements are formalized in an executable *model*. The model is built with AutoFocus [7], a model based CASE tool with a time synchronous operational semantics. It is based on concurrent components communicating over directional typed channels. A component may be composed of subcomponents or an extended finite state machine.

If inconsistencies in the requirements are found when modeling the system, the requirements have to be clarified in coordination with their stakeholders. The executable model, implementing the clarified requirements, can be translated into several other languages. In this approach theories for the theorem prover Isabelle (see Section 2.3) and C code can be generated.

In addition to the model, system parameters for the operating system and the communication network, e.g. the scheduling tables, are specified. These are required by the code generators and the formal verification.

In this phase only a model of the application is built, operating system, network and hardware are not regarded. The synchronous model is later on deployed onto a time-triggered platform, which still maintains the timing properties of the system. In this way the system remains time synchronous throughout development and run time (see also Section 3) as shown in [2].

2.3 Verification

For the verification of the system the interactive higher-order general purpose theorem prover Isabelle [9] is used. It allows the system to be decomposed into *theories*, which are roughly the equivalent of modules in procedural programming languages.

Once the Isabelle theories have been generated and the properties have been formalized, the verification work can start. Theories for the OSEKtime operating system and the FlexRay communication network have been formalized and verified as parts of the framework (see Section 3). These theories and their properties can be reused for other applications, so only the properties of the application itself have to be shown.

The subsystems that have been modelled separately in the design phase can also be verified separately. The case study (see Section 4) showed that the verification of smaller subsystems is easier than verifying the system as a whole. After the verification of the subsystems these can be integrated. The verification of this integrated system becomes easier, since the subsystems now provide a wide range of proven properties, which can be used. Another benefit is that these verified subsystems can be reused for other systems. By proving the correctness of the integrated system, the integration itself is shown to be correct.

Since the FlexRay communication system provides fault containment for network errors, verified and non-verified ECUs can be integrated into the same network, without jeopardizing the correctness of the verified components.

If there are errors detected during the verification phase, the model and/or requirements have to be modified to be able to fulfill the properties. Then new theories can be generated from the model and the verification process can continue. In the presented approach the generated Isabelle theories are proven since this is more effective than verifying the generated C code. By proving the equivalence of model, theories and C code using translation validation [13]

the properties that have been verified for the Isabelle theories also hold for the C code. This proves that the system running as C code on an ECU really satisfies the properties specified in the requirements phase.

2.4 Integration

Once all properties are proved to be valid for the model, C code can be generated from it. This code can then be compiled for and executed on the target platform.

To be able to argue about the real time properties and schedulability of the operating system tasks, an upper bound for the worst-case execution times (WCET) can be estimated for each task. For this measurement the C code and the characteristics of the microprocessor of the ECU are required. For the WCET analysis existing tools can be used, which are not part of this work. Once the WCETs have been estimated, it can be verified if they, together with the operating system schedule, meet the real time requirements. Together with the verification described above this guarantees system correctness.

3. VERIFICATION FRAMEWORK

The verification framework provides the methodology for the verification of single subsystems, as well as for their integration. The framework is oriented towards the *time-triggered paradigm* (time-triggered communication protocol and time-triggered operating system) because it provides deterministic time behavior, which is important for distributed real time applications. In a time-triggered system all actions are executed at predefined points in time. An exact time schedule is a good basis for the deterministic system behavior because of fixed task execution times and order, as well as deterministic message transmission times.

3.1 Architecture

The developed verification framework is split into layers that depict the typical technical infrastructure of ECUs: hardware, communication protocol, operating system, and application. This structure is shown in Figure 2. For the framework FlexRay (see [1] and [5]) was chosen as a communication protocol and OSEKtime [12] as operating system, but the main ideas are also applicable to other time-triggered communication protocols and operating systems.

FlexRay is a communication protocol for safety critical real time automotive applications that has been developed by the FlexRay Consortium [4]. It is a static time division multiplexing network protocol and supports fault-tolerant clock synchronization via a global time base.

OSEKtime OS (Time-Triggered Operating System) is an OSEK/VDX [10] open operating system standard of the European automotive industry. The OSEKtime OS supports static cyclic scheduling based on a computation of tasks' WCETs (worst case execution times). FTCom [11] is an OSEKtime fault-tolerant communication layer that provides a number of primitives for interprocess communication and makes task distribution transparent.

The integrated system is modeled as a network of nodes that are connected by a FlexRay bus. Every node runs a number of application processes and an operating system. To maintain synchrony of the distributed application processes on different nodes the cycle length of the scheduling tables has to be the same for all the nodes.

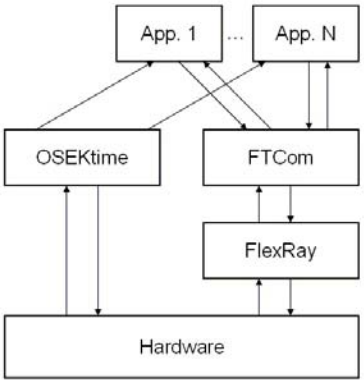

Figure 2: Verification Framework Architecture

The syntactical compatibility of applications interfaces is provided in the design phase by the AutoFocus [7]. The semantical compatibility of the layers is proven in the verification phase by verification of the properties.

3.2 Verification

The developed verification framework is oriented towards the integration of independently developed subsystems into a sound system. The overall verification task is decomposed in two dimensions: verification of layers, available in the developed framework, and verification of application processes. While the correctness of the application processes has to be shown separately for every application (see also Section 4), the correctness of the underlying layers is shown once and can be reused. The verification of all the layers yields the verified system. When proving the properties of the applications it is necessary to assume that the operating system, the hardware and the bus connecting the nodes behave correctly. The following list provides an excerpt of the properties proven for FlexRay and OSEKtime and that are required by the applications:

- The behavior of the FlexRay and OSEKtime schedulers is cyclic, which implies deterministic behavior.
- The messages received by the FlexRay controller are not lost or modified.
- If a message is send by a node, it will be received by all the other nodes.
- Every message identifier in FTCom message list is unique and no FTCom function can destroy this property. This guarantees that on arrival of a new message the old message of the same type is overwritten.
- The OSEKtime scheduler does not change any application data.
- The application processes are called by OSEKtime on time and in a correct way (according to the dispatcher table and the scheduling policy).

These properties are used in the verification of the case study, which is presented in the next section.

4. CASE STUDY

The goal of the case study was to demonstrate feasibility of the introduced approach. The verification activities presented in this paper aim to prove the correctness of an *emergency call system*. According to the plans of the European Union and European automotive industry all new cars in Europe will be equipped with automatic emergency call (eCall) technology by 2009 [3]. The system's purpose is to notify the rescue services in the case of an accident. Thereby the coordinates of the vehicle are transmitted. Afterward it has to try to establish a voice communication between a call center operator and the driver. The system exhibits some typical properties of the automotive domain, such as *real-time requirements*, *safety requirements* and *distribution*.

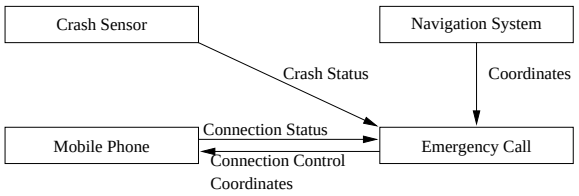

Figure 3: Interaction Dependencies between Subsystems

The emergency call system consists of a network of subsystems, visualized by Figure 3. The central subsystem, responsible for the execution of the emergency call, has to interoperate with a number of further subsystems in order to deliver the demanded functionality. In particular the *crash sensor* activates the emergency call sequence by detecting an accident. The *navigation system* delivers the current coordinates of the vehicle and the *mobile phone system* handles the actual call. These dependencies lead to the fact that the correctness of the emergency call functionality is dependant on the correct functioning of the listed subsystems, as well as the correctness of their interactions. The further prerequisite is the aforementioned correctness of the execution environment.

The execution environment, the system is deployed to, consists of two ECUs connected by a FlexRay link (see Figure 4). It is modelled as an instance of the layered reference architecture presented in Section 3. The subsystems crash sensor and navigation are deployed to the ECU 1; emergency call and mobile phone – to ECU 2.

The properties which have been proven for the systems are:

- The emergency call is not initiated as long as there is no crash.
- The emergency call is performed within the specified time after the crash.

The verification of the first item is simple: it is sufficient to verify that the emergency call application remains in the same inactive state as long as no crash signal is received from the sensor. The verification of the second item is much more involved because it is necessary to verify the correct interaction between the mobile phone and the emergency call. Each application-level process was modelled as an automaton and the verification task was to prove that the automatons cooperate properly. The Isabelle theorems proved

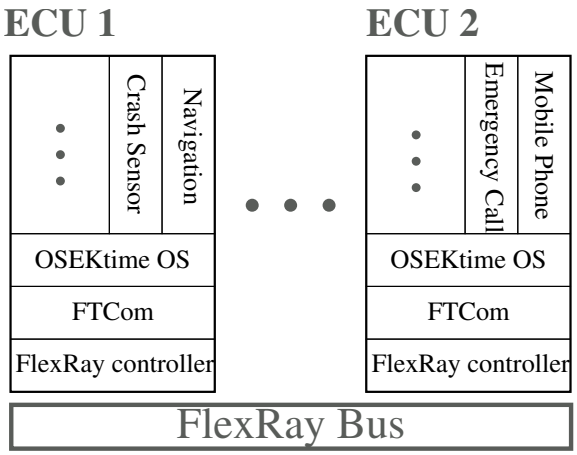

Figure 4: Deployment Architecture of Emergency Call System

for the emergency call system rely on the verified properties of its run-time environment (e.g. of OSEKtime and FlexRay) discussed in Section 3. The theorems are not presented here because the paper aims to emphasize the verification principles and not the technical details.

The result of the case study was a provably correct integrated emergency call system which guarantees the demanded behavior. The study demonstrated applicability of the verification framework within the automotive domain.

5. RELATED WORK & CONTRIBUTION

The idea to use formal verification as quality assurance means for safety critical systems is not new and the question what is special in the presented work is legitimate.

Verification of general purpose operating systems (or some aspects of these systems) is for example addressed by Mark Hillebrand [6] and Sergey Tverdyshev [16]. These works deal with verification of some aspects of the L4 microkernel [8], which can provide a basis for large scale operating system verification. Although important on their own, these works are not directly applicable to time-triggered systems. The conceptual core of time-triggered systems is a firm schedule that makes the system behavior deterministic. Thus, a key requirement when verifying a real time system is the correctness of the scheduler, as well as of the schedule itself. These issues are not addressed in the works cited above.

Verification of time-triggered systems was addressed by John Rushby in [14]. However, Rushby addresses verification of bus protocols only. Although important on its own, verification of bus protocols is not sufficient to build a verified time-triggered system. To build a complete verified system, it is necessary to add at least a verified time-triggered operating system and a verified application.

As compared to other verification work, the presented approach has two important distinguishing features:

- It treats the system as a whole, especially taking module interfaces into consideration. This also requires structuring of the verification tasks and implies dependencies of verification properties. In the presented work the dependencies of the verification tasks were implied by the layered system structure.

- The presented work treats time-triggered systems. Time-triggered systems are different form the event triggered ones in that they require other properties to be verified, namely correct schedules and correct schedule execution.

To sum up, the presented approach reaches the verification of the integrated system, which is new in the time-triggered domain.

The verification of the whole system bases on the verification of a standard time-triggered operating system (OSEKtime) and a bus protocol (FlexRay). They were formalized as a prerequisite for the case study. To obtain a verified integrated system, the infrastructure (OS and bus) properties that are necessary for application verification were verified.

An additional result of the presented work is the demonstration how model-based development can be applied in the practice. The idea of model-based development, as introduced by Schätz et. al [15], is to build an explicit model before going to code. The development starts with a simple model that is enriched and refined during the development. The final result of the whole development process is code that is obtained by successive model refinement. Model-based development of synchronous systems deployed onto time-triggered platforms was already approached for example by Caspi et. al in [2], but they did not perform any verification. For the verification purpose it means that certain properties can be shown for the model first and then it must be proven that the refinement steps do not invalidate the properties. This way of successive model refinement can be followed using the presented approach.

6. CONCLUSION

The goal of the presented work was to demonstrate the feasibility of formal verification in automotive software development. The feasibility was shown in two steps: Introduction of a framework for verification of automotive software and validation of the framework on a case study. In order that the framework itself be sensible its single constituents (OSEKtime scheduler, OSEKtime communication layer (FTCom), and FlexRay) were verified. Properties to be verified for the standard modules were identified on the basis of the infrastructure requirements posed by the application. These infrastructure requirements were specific to the time-triggered domain, which provides reusability of the verified properties.

The application properties to be verified were identified on the requirements level. Then they were verified under assumptions that had already been proven for the infrastructure. This modular verification approach resulted in a verified modular system and circumvented the unmanageable "all–in–one" verification. The performed case study showed feasibility of this approach. Last but not least, the presented work was motivated and supported by BMW.

7. FUTURE WORK

The presented work demonstrated that verification of automotive software is feasible. However, feasibility alone is not sufficient for the method to become accepted by car manufacturers. It is also the question of the development cost that is vital. Thus, quantifying the verification efforts can be an interesting direction of further research. To perform these measurements a new case study shall be performed. This case study will also include harder real time requirements. The introduced verification framework can also be made more adequate by including explicit fault models, typical in the automotive domain, and verification of fault tolerance. Hard real time requirements and fault tolerance verification would make the results of the case study and of the effort quantification better applicable to the OEM development process.

8. ACKNOWLEDGMENTS

We would like to thank Dr. Oscar Slotosch, Dr. Katharina Spies, and Prof. Manfred Broy at the Technische Universität München as well as Richard Bogenberger and David Trachtenherz at BMW Research.

9. REFERENCES

[1] FlexRay Consortium . *FlexRay Communication System - Protocol Specification - Version 2.0*, 2004.

[2] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications. In *LCTES '03: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 153–162. ACM Press, 2003.

[3] European Commission (DG Enterprise and DG Information Society). eSafety forum: Summary report 2003, March 2003.

[4] FlexRay Consortium. `http://www.flexray.com`.

[5] FlexRay Consortium. *FlexRay Communication System - Bus Guardian Specification - Version 2.0*, 2004.

[6] M. Hillebrand. *Address Spaces and Virtual Memory: Specification, Implementation, and Correctness*. PhD thesis, 2005.

[7] F. Huber, B. Schätz, and G. Einert. Consistent Graphical Specification of Distributed Systems. In J. Fitzgerald, C. Jones, and P. Lucas, editors, *Industrial Applications and Strengthened Foundations of Formal Methods (FME'97)*, LNCS 1313, pages 122–141. Springer Verlag, 1997.

[8] L4 microkernel. `http://os.inf.tu-dresden.de/L4/`.

[9] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[10] OSEK/VDX. `http://www.osek-vdx.org`.

[11] OSEK/VDX. *Fault-Tolerant Communication - Specification 1.0*, 2001.

[12] OSEK/VDX. *Time-Triggered Operating System - Specification 1.0*, 2001.

[13] A. Pnueli, M. Siegel, and E. Singerman. Translation validation. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, page 151. Weizmann Institute of Science, Rehovot, Israel, 1998.

[14] J. Rushby. An overview of formal verification for the time-triggered architecture. In W. Damm and E.-R. Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*, pages 83–105, Oldenburg, Germany, Sept. 2002. Springer-Verlag.

[15] B. Schätz, A. Pretschner, F. Huber, and J. Philipps. Model–based development of embedded systems. In J.-M. Bruel and Z. Bellahsene, editors, *Advances in Object-Oriented Information Systems*. Springer, 2002.

[16] S. Tverdyshev. Documentation and modelling of the ipc mechanism in the l4 kernel. Master's thesis, 2003.

[17] Verisoft Project. `http://www.verisoft.de/`.