Development of Software for Open Autonomous Automotive Systems in the *Stadtpilot*-Project

Andreas Reschka, Jürgen Rüdiger Böhmer
Institute of Physics and Technology
Universität Hildesheim
Hildesheim, Germany
{andreas.reschka, boehmer}@uni-hildesheim.de
+49 (0) 5121 883-923

Jan Gačnik, Frank Köster
Institute of Transportation Systems
German Aerospace Center (DLR)
Braunschweig, Germany
{jan.gacnik, frank.koester}@dlr.de
+49 (0) 531 295-3437

Jörn Marten Wille, Markus Maurer
Institute of Control Engineering
Technische Universität Braunschweig
Braunschweig, Germany
{wille, maurer}@ifr.ing.tu-bs.de
+49 (0) 531 391-3886

***Abstract*— Following the successful participation in the DARPA Urban Challenge with the *CarOLO*-Project [1], the *Stadtpilot*-Project [2] aims at implementing and evaluating autonomous driving with passenger vehicles in regular urban traffic on the inner city ring road of Braunschweig. Transferring autonomous driving from an artificial urban environment as in the Urban Challenge[1] in 2007 to regular urban traffic has a major impact on the quality requirements for both hardware and software systems to be developed.**

This paper focuses on the development process of the safety-critical software – including functions, architecture and tests for prototyping open autonomous systems in urban traffic. The key issue is to establish flexibility in development and openness for Car-to-Car (C2C) and Car-to-Infrastructure (C2I) communication while still guaranteeing a high level of safety. This paper points out new techniques and lessons learned used with respect to the current standards in the field of functional safety for automotive systems (esp. ISO/DIS 26262 [3]).

***Index Terms*— Autonomous Driving, Urban Mobility, Functional Safety, Software Engineering.**

I. Introduction

The *Stadtpilot*-Project aims at driving autonomously along the inner city ring road of Braunschweig, Germany, with a length of 11 km (Fig. 1). Autonomous vehicles have to perform complex driving maneuvers like merging into moving traffic, driving on two-lane roads, performing lane changes, and passing intersections according to traffic regulations. In October 2010 the *Stadtpilot*-Project demonstrated first public rides on a north-eastern part of the ring road[2], which is a representative part of an urban environment. The autonomous vehicle was able to follow a given trajectory along its lane, to pursue other road users with a safety margin and to stop at traffic lights. Also a u-turn on an intersection in the middle of the track was done autonomously.

For this challenging task, the project team has set up a modified Volkswagen Passat station wagon called *Leonie* (Fig. 2). It is equipped with an additional generator, a full set of automotive sensors, a server rack, and an interface to the relevant actuators to fulfill all the requirements of an autonomous vehicle for urban environments. A second vehicle called *Henry* is under development and will be equipped in a similar way.

However, most of the complex functionality is expressed in terms of software. For the demonstration in 2010, about 130,000 lines of C++ code were involved in the autonomous driving on public roads. Achieving high quality of this code is a complex task as well,

[1] http://www.darpa.mil/grandchallenge/

[2] Press Information 134/2010 from TU Braunschweig: http://tinyurl.com/37yqxga

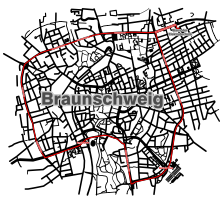

Fig. 1. Map of the inner city ring road of Braunschweig.

Fig. 2. The autonomous vehicle Leonie.

especially with respect to *safety*, *flexibility* and *openness*, and demands the use of state-of-the-art software engineering methods.

This paper shortly describes the architecture of the *Stadtpilot* software, presents the way how the *Stadtpilot* team copes with software development, resumes insights into lessons learned and sketches future changes to the software engineering process to achieve even better results. The whole process is derived from the approach taken in the *CarOLO*-Project[4].

II. Architecture

The overall software architecture (Fig. 3) is strongly modularized.

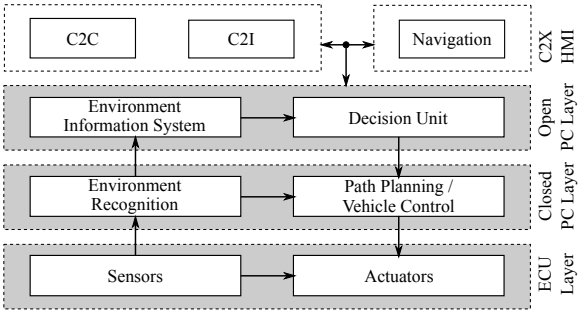

Fig. 3. Simplified *Stadtpilot*-Architecture [2].

Each major component can be exchanged to test, compare and benchmark different approaches and thus guarantees a high flexibility to developers. For the projects KOLINE[3] [5] and AIM[4] an interface for C2C and C2I technology is integrated and will be used for further improvements.

The following section describes three layers with different jobs in the overall system and therefore slightly different requirements.

[3]Cooperative and Optimised Traffic Signalling in Urban Networks (KOLINE): http://www.projekt-koline.de/

[4]Application Platform for Intelligent Mobility (AIM): http://tinyurl.com/3ymmbtd

A. ECU Layer

The Sensors and Actuators are controlled by electronic control units (ECUs). This allows fail-safe operation for low-level functions like braking, accelerating and steering. The interface to the actuators is a CAN bus gateway that connects the vehicle CANs to an additional CAN bus used in the *Stadtpilot* PCs. Some sensors are connected via CAN as well and the others are connected via Ethernet.

B. Closed PC Layer

The PCs in this layer handle the sensor data and do the low-level *Path Planning* and *Vehicle Control*. In *Enviroment Recognition* sensor data is collected and partly analyzed to generate object lists or occupancy grids as a basis for driving decisions. The complexity of these tasks is another main reason for the usage of PCs. The *Path Planning* component computes a trajectory which the vehicle should follow and the *Vehicle Control* contains controllers for the actuators. All of these components communicate in real-time using RTI DDS[5] with well-defined interfaces and can be exchanged easily.

C. Open PC Layer

This layer collects information from the *Stadtpilot* system and provides an interface to external systems. It is also responsible for making high-level driving decisions like lane changes and turns. C2C and C2I applications will be integrated into this layer thus can provide further information for decision making.

Additionally an HMI system has been developed for the co-driver to monitor Leonie's state. This involves the actual position on high-resolution aerial photos, the position of recognized objects and some vital information like the current positioning reliability. A text-to-speech system informs the safety driver and the co-driver about the actual system state. Furthermore, this HMI fulfills requirements derived from the homologation documents.

III. Requirements for Software Development

Sufficient high quality of software can only be achieved through a structured and easy-to-apply development process. Advanced tool support is necessary to help developers manage complexity. Besides software quality, system robustness, traceability

[5]Real-Time Innovations Data Distribution Service: http://www.rti.com/products/dds/index.html

regarding requirements, implementations, test cases and documentation are major concerns.

A. Safety

When driving autonomously on the city ring road of Braunschweig, safety is the most important issue. To be allowed to drive autonomously on public roads, a homologation procedure for *Leonie* was done, involving public authorities, including a safety concept. The major requirements derived from this safety concept are a structured development process which is described in this paper, intensive testing, process documentation, HMI and a safety driver who is able to gain control over the vehicle in every situation by applying the brake or accelerator pedal, by turning the steering wheel, changing the gear or activating the electric parking brake. This intervention is meant to be the last safety mechanism in case of a total system failure.

When thinking about taking autonomous vehicles or at least part functions into mass market one day, even more has to be considered, especially from the product liability point of view. This is where safety standards as IEC 61508 [6] and ISO/DIS 26262 [3] are getting involved.

B. Flexibility

The *Stadtpilot*-Project is a long-term research project, involving several partners who develop software for the vehicles. Thus, there is a high demand for adding or modifying software. Another important feature is changing between different software binary versions for benchmarking different approaches which applies to development of driver assistance systems for the mass market as well and demands a safe mechanism to install updates and bug-fixes. For instance, OSGi [7] is used in several comfort-oriented applications in the market, while current usage in real-time applications is still limited [8].

C. Openness

In upcoming scenarios, *Leonie* will also be used for cooperative applications. This involves using C2C and C2I communication. Using C2C technology, especially cooperative driving [9] with other research vehicles is a planned application. With respect to the use of C2I, the research projects KOLINE and AIM will feature interaction of *Leonie* with roadside unit based communication, involving information about the state of traffic lights. This is primarily used to demonstrate the capabilities to improve the efficiency of automotive transportation in urban traffic.

For further research and development, opening architectures becomes a major issue by applying C2C and C2I communication. Many new business models and applications [10], [11] become feasible, including different business partners beyond the automotive manufacturer. To cope with bigger develpoment teams, more software modules and more complex interfaces the following development process will help to improve quality of code and speed of development.

IV. Software Development Process

The process framework in this iterative approach is derived from ISO/DIS 26262, shown in Fig. 4. Small-scale development is then based on the Scrum methodology [12]. In every iteration, a new or improved driving function (e.g. autonomous lane change) is introduced to the vehicle. Beginning with the capturing of requirements, an impact analysis on the existing architecture is carried out and software units are designed. After their implementation, unit and system tests are run, and the new function is tested on a closed test track with other vehicles and obstacles according to a well-defined procedure. With fulfillment of all steps in the process, the new function is deployed into operation in the urban environment.

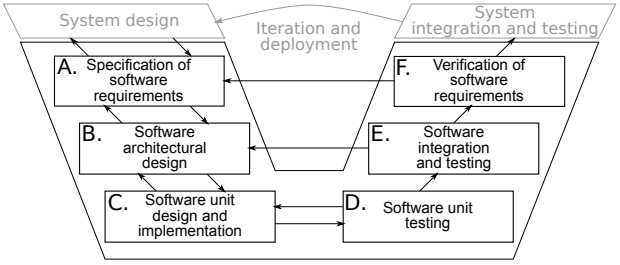

Fig. 4. Software Development Process, adapted from ISO 26262, part 6.

The strong focus on the safety of road users is the main aspect of autonomous driving in a real urban environment. For safety-critical components the development process includes intense unit and system tests, and it meets both ISO/DIS 26262 recommendations and homologation requirements of the TÜV.

A. Specification of Software Requirements

In order to build an autonomous driving system the requirements for this system are defined by the situations encountered by the vehicle in order to drive safely in its environment. In the *Stadtpilot*-Project

an urban environment provides many complex scenarios. To solve these scenarios, the abilities of the vehicle are extended iteratively in short development cycles. For every new situation the vehicle is able to handle, the requirements are defined and the consequences to the overall system are derived. Therefore a milestone in the Trac[6] and Agilo[7] system is defined, e.g., "Reaction on transmitted state information from traffic lights". Each involved developer creates one ore more *User stories* which describe functions necessary for the milestone from the developers domain. Every user story is described more detailed in tickets which contain a short work product and a showcase the developer has to present to the team after finishing a ticket. A user story is finished if all tickets are implemented and tested succesfully. With the usage of Trac and Agilo many errors can be prevented in this early stage and the whole process is well documented. Additionally, test scenarios for a closed test track are derived from the defined driving situations and committed to all developers.

B. Software architectural design

Within each iteration in the development process the architecture is adapted and extended to fulfill the new requirements defined in the preceeding step. Until now, no major changes to the architecture were necessary, but in every step an extension to the architecture was done. With the use of UML Tools this step is documented and design decisions can be traced easily, e.g., new modules and interfaces between new and existing modules are defined and commited to the Subversion[8] system.

C. Software unit design and implementation

All components are designed and implemented in an agile development process. In the first step interfaces and internal functions are designed and specified in short developer meetings.

After the design process, the components are implemented as so-called filters within the development platform ADTF (Automotive Data and Time-triggered Framework) from Electrobit[9]. To shorten the development cycle in comparison to serial development and to allow early testing and benchmarking for different algorithms and functions, these filters are implemented as prototypes and tested as soon as possible. To improve the quality of the resulting code standard methods like coding guidelines and code analyzing tools are applied. With Doxygen[10] the code gets documented simultaneously.

The target hardware in the vehicle are standard industry PCs and run the Debian[11] operating system with the Linux preemptive real-time patch[12].

D. Software unit testing

Unit testing is done using manual test cases as well as generated ones, in detail described in [13]. With the use of state-of-the-art methods for C++ unit testing and the testing capabilities integrated in ADTF and the use of continuous integration tools like buildbot[13], the test cases and the unit testing process is highly integrated in the development process. Every committed change to the source code triggers an automated build and test procedure for the whole system to check for bugs and broken dependencies.

E. Software integration and testing

Overall testing of software functionality and architecture is done primarily on a simulation system in ADTF and then on an isolated test track for each software release. In the simulation the system behavior is visualized and tested in desired traffic situations. Currently a more powerful simulation environment is under development.

If the simulation succeeds the software is applied to the vehicle PCs and basic functionality like activating and deactivating the system, following a course and safety driver interaction are tested on an empty test track.

After fulfilling all basic test cases the creation of software packages for each involved PC is used to save software versions that can be used to test the functional behavior in real urban traffic. Through the use of packaging software versions, it is easily possible to switch between different software releases for the same hardware configuration. Those releases can have different driving abilities and can fit into different environments.

The packaging process (Fig. 5) is based on the Debian packaging system, which has extensive and rich dependency information used in a variety of applications [14]. At first calibration data from ADTF and

[6] http://trac.edgewall.org/
[7] http://www.agile42.com/cms/pages/agilo/
[8] http://subversion.tigris.org/
[9] ADTF Blog: http://www.eb-assist-blog.com/eb-assist-adtf/
[10] http://www.stack.nl/ dimitri/doxygen/
[11] http://www.debian.org/
[12] http://rt.wiki.kernel.org/
[13] http://trac.buildbot.net/

the configured software in an Executable and Linkable Format (ELF) is verified and then integrated into installable Debian packages for each vehicle PC. After installation this application-specific software is verified again and then ready to use.

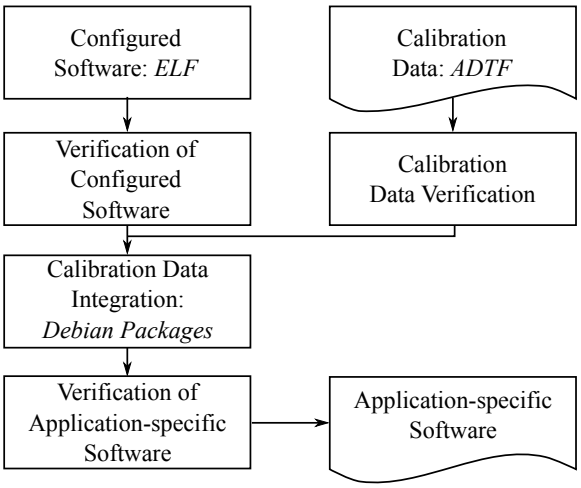

Fig. 5. Software configuration and packaging process, adopted from ISO 26262, part 6, appendix C.

This allows a very fast and flexible exchange for different software releases, maintaining the consistency on the system based on dependency models. Also a mechanism similar to the OSGi [7] process used for vehicle telematics applications is possible.

F. Verification of software requirements

After successful simulation and basic testing on the test track, more complex driving scenarios on an urban traffic like test track are performed, involving additional vehicles, other obstacles and traffic lights. The execution of these driving maneuvers guarantees the system behavior and informs the safety driver of how the vehicle acts and reacts in different situations. To prevent a too extensive test process, the desired scenarios are selected according to software and component changes before testing starts. To automate the scenario selection process, advanced analysis methods [15] can be utilized in the future for impact analysis and for deriving test cases from functional scenarios.

G. Iteration and Deployment

The last step in the development process are rides in real urban traffic on the inner ring road of Braunschweig with installed and packaged software versions. Therefore a track is created on which *Leonie* has to drive autonomously. This track can either be a part or the entire ring road. At the moment the course is calculated off-line, in further development a strategic planning unit calculates the best course from the actual position to a desired position on the ring road similar to a navigation system.

After successful driving on the city ring, another iteration cycle begins, restarting the process. The software packages derived during software integration and testing are archived and thus can be redeployed later. This can be used for comparison of a new software state with an older state with respect to controller performance.

H. Summary

The process is iterative and for each new scenario that is added to the abilities of the vehicle the whole process starts again to ensure a complete definition, realization and testing procedure of software units and the overall system. It focuses on providing sufficient levels of flexibility and safety for developing and testing autonomous systems in urban traffic scenarios. Furthermore, the *Stadtpilot*-Project aims at long-term developments and evaluations. Another major challenge is handling open systems, as future autonomous systems are likely to be massive cooperative systems [9], making use of C2C and C2I communication. Besides computing power, this is one of the major reasons for using PCs instead of ECU hardware.

V. Related Work

Model-based development (MBD) [16] is a major approach that many car manufacturers and even the NASA [17] follow. Developers can build their systems with easy-to-learn modeling tools for requirements, architecture, software components and test cases. The resulting models abstract the complex functionality of ECUs for control and autonomous systems and allow a full view on the system architecture. With usage of code-generators, the models, e.g., from Matlab/Simulink[14] can be transferred to C++-Code.

Although MBD is a state-of-the-art approach, development without intensive model generation is possible by applying classical Software Engineering methods like *separation of concerns* and *structuring*. Currently, those methods are used in the *Stadtpilot*-Project and the software components are implemented manually as ADTF-Filters.

[14]MathWorks©Homepage: http://www.mathworks.com/

VI. Lessons Learned / Conclusions

The described process refines the exceptional TÜV homologation and admits the *Stadtpilot*-Project to legally drive autonomously in real world traffic in Braunschweig with reasonable testing effort. The open architecture of the systems allows a high level of flexibility to developers and guarantees compatibility to further developments in C2C and C2I technology.

While creating the described development process many aspects from ISO/DIS 26262 and IEC 61508 were very helpful. Thus, at some development stages more effort is taken than formally required by the homologation. This involves integration of tools for code reviews, code analysis, and coding guidelines in the developed process. In contrast, not the whole set of requirements for serial development from the standards is currently addressed, e.g., no full risk, hazard and fault-tree analysis.

The described process allows benchmarking different software components in real world situations, and improving them. It also permits finding vulnerabilities and key issues in the field of autonomous driving in urban traffic.

With the use of standard software and hardware a clean configuration management was established. The best results could be achieved with the Debian operating system with additional real-time support and the usage of the native package management for managing software releases.

Cooperative driving is a way to reduce the probability of accidents and to increase the efficiency of traffic. This is enabled by the use of C2C and C2I communication and introduces new challenges regarding safety and especially security.

In future development more complex traffic scenarios will require an even better reliabilty of driving decisions in higher levels of the architecture. Therefore the development process hast to be evolved and maybe more tools will be integrated in the toolchain for the development process to improve software quality and reduce development cycle times.

References

[1] C. Basarke, C. Berger, K. Berger, K. Cornelsen, M. Doering, J. Effertz, T. Form, T. Gülke, F. Graefe, P. Hecker, K. Homeier, F. Klose, C. Lipski, M. Magnor, J. Morgenroth, T. Nothdurft, S. Ohl, F. Rauskolb, B. Rumpe, W. Schumacher, J. M. Wille, and L. Wolf, "Caroline: An autonomoulsy driving vehicle for urban environments.," *Journal of Field Robotics*, vol. 25, no. 9, pp. 674–724, 2008.

[2] J. M. Wille, F. Saust, and M. Maurer, "Stadtpilot: Driving autonomously on braunschweig's inner ring road," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, jun. 2010, pp. 506 –511.

[3] International Organization for Standardization (ISO), *ISO/DIS 26262: Road vehicles – Functional safety* , 2009.

[4] C. Basarke, C. Berger, and B. Rumpe, "Software & systems engineering process and tools for the development of autonomous driving intelligence.," *Journal of Aerospace Computing, Information, and Communication (JACIC)*, vol. 4, no. 12, pp. 1158–1174, October 2008.

[5] F. Saust, J. M. Wille, O. Bley, R. Kutzner, B. Friedrich, and M. Maurer, "Exploitability of vehicle related sensor data in cooperative systems," in *IEEE Proceedings of the International Conference on Intelligent Transportation Systems*, 2010.

[6] International Electrotechnical Commission (IEC), *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2005.

[7] Yuantao Li, F. Wang, Feng He, and Z. Li, "Osgi-based service gateway architecture for intelligent automobiles," in *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, June 2005, pp. 861 – 865.

[8] T. Richardson, A. J. Wellings, J. A. Dianes, and M. Díaz, "Providing temporal isolation in the osgi framework," in *JTRES '09: Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems*, New York, NY, USA, 2009, pp. 1–10, ACM.

[9] M. Röckl, K. Frank, T. Strang, M. Kranz, J. Gačnik, and J. Schomerus, "Hybrid Fusion Approach combining Autonomous and Cooperative Detection and Ranging methods for Situation-aware Driver Assistance Systems," in *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (CFP08PIM-CDR)*. 2008, IEEE.

[10] CAR 2 CAR Communication Consortium, *Manifesto*, August 2007.

[11] European Telecommunications Standards Institute (ETSI), *ETSI TR 102 638 – Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions*, v1.1.1 edition, June 2009.

[12] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[13] F. Saust, T. Müller, J. M. Wille, and M. Maurer, " Entwicklungsbegleitendes Simulations- und Testkonzept für autonome Fahrzeuge in städtischen Umgebungen ," in *AAET*, 2009.

[14] S. Spaeth, M. Stuermer, S. Haefliger, and G. von Krogh, "Sampling in open source software development: The case for using the debian gnu/linux distribution," in *2007. HICSS 2007. 40th Annual Hawaii International Conference on System Sciences*, jan. 2007, pp. 166a –166a.

[15] J. Gačnik, "Providing Guidance In An Interdisciplinary Model-Based Design Process," in *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 2010, pp. 130 – 137, IEEE Computer Society.

[16] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering*, New York, NY, USA, 2006, ICSE '06, pp. 33–42, ACM.

[17] J. Schumann and W. Visser, "Autonomy software: V&V challenges and characteristics," in *Aerospace Conference, 2006 IEEE*, 2006, pp. 1 –6.