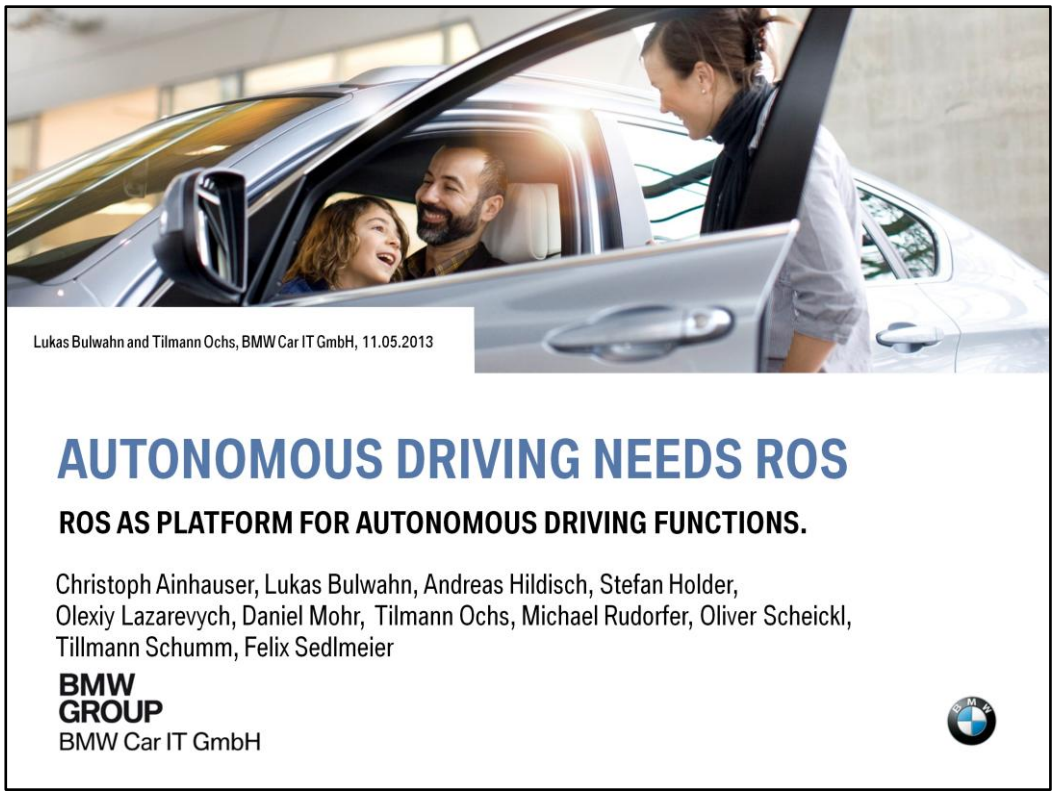
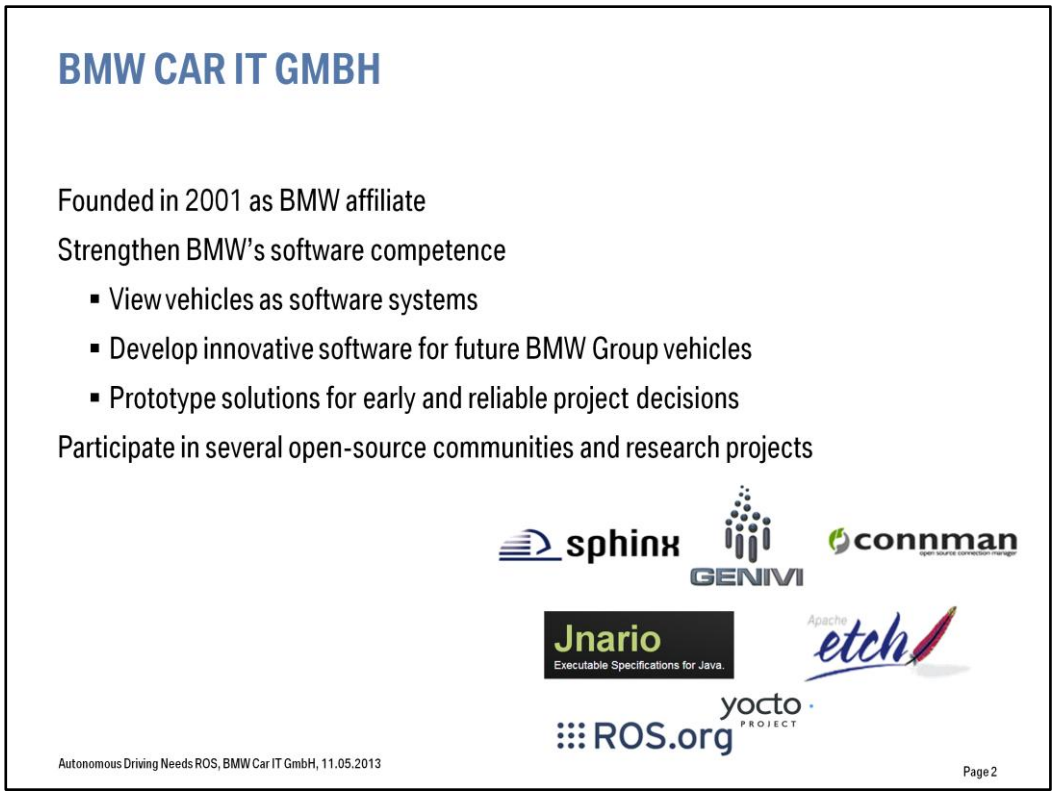


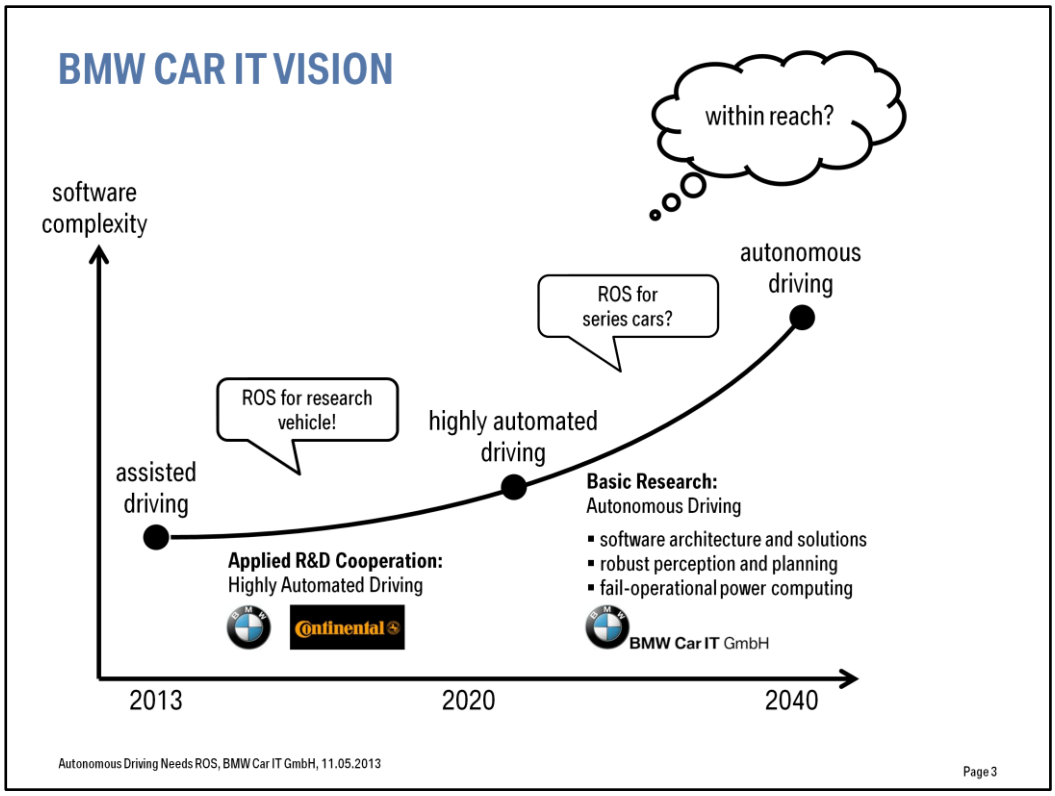
The presentation discusses if and to what extent ROS can serve as a platform for future autonomous driving functions.
It presents ideas from joint work of several people at BMW Car IT: Christoph Ainhauser, Lukas Bulwahn, Andreas Hildisch, Stefan Holder,
Olexiy Lazarevych, Daniel Mohr, Tilmann Ochs, Michael Rudorfer, Oliver Scheickl, Tillmann Schumm, Felix Sedlmeier



BMW Car IT in Munich, Germany, is strongly connected to the car manufacturer BMW.
It was founded in 2001 as a full affiliate of BMW, and serves as a think tank and a research department for BMW.
Our company‘s focus is on software development and software engineering for future cars and our main task is to strengthen BMW‘s software competence.

At BMW Car IT, we view a vehicle as a connected, distributed network of complex software systems. Mainly, we create innovations for future vehicles by building software prototypes and new software solutions. The prototypes provide early and reliable project decisions for the departments working on the series development.

Open source software not only promotes innovation, open source software accelerates progress in software development. Active engagement in open-source communities is a key contribution for us to build high-quality software with short time-to-market cycles. A selection of open-source projects we maintain and regularly contribute to are GENIVI, connman, Apache Etch and the Yocto Project. In the ROS community, we work on a small project to build a Yocto Layer for ROS.



This graph shows our current roadmap for increasing automation of driving activities.

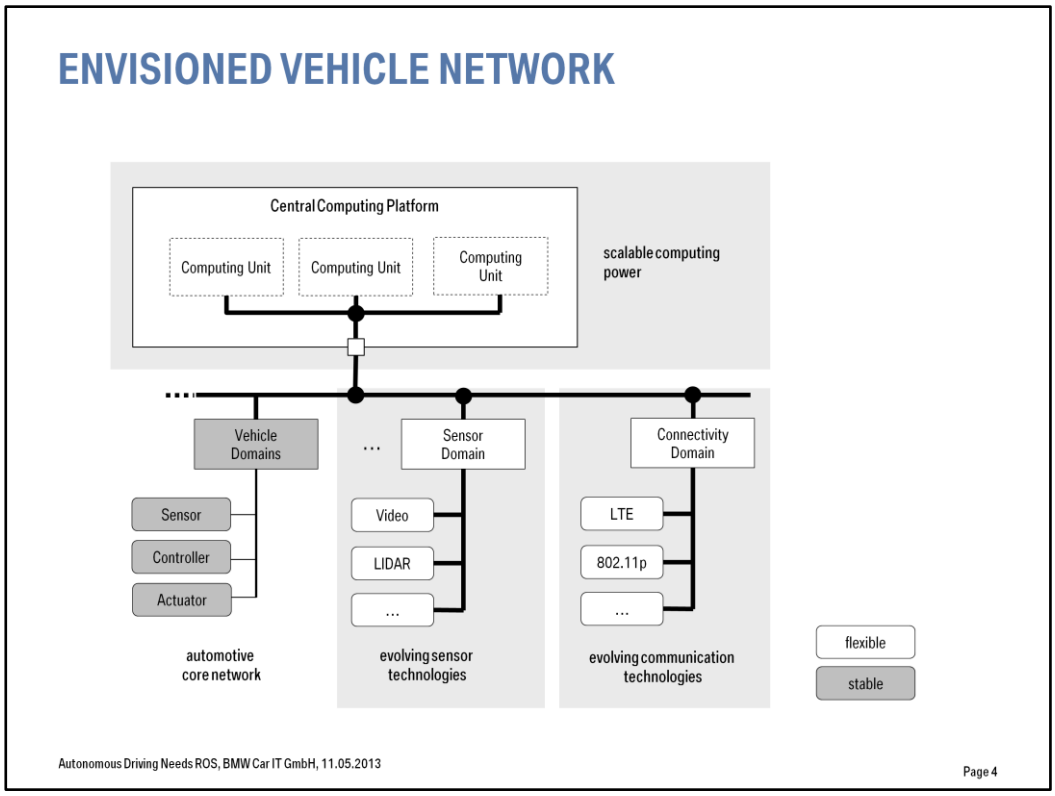
Currently, modern series cars have integrated assisted driving functions, such as adaptive cruise control and parking assistants. In the future, this sort of driving automation will grow in its functionality, but of course also, in its software complexity and safety requirements.

In ten years, we expect that modern cars can drive in some situations, like on freeways, with only minor control from the driver. This is termed highly automated driving.
In thirty years time, we imagine that modern cars drive in every-day situations, allowing the driver to conduct the vehicle only if he wishes to. This is termed autonomous driving.

There are many engineering tasks and research questions that have to be solved for this vision. For highly automated driving, the challenges are to obtain a suitable environment model around the car with affordable sensors, to predict movements of other cars on the freeway, and to reliably observe the driver to ensure that she can take over whenever it is neccessary. These challenges are currently addressed by the applied R&D cooperation between BMW and Continental in the Highly Automated Driving project. This project is planning to use ROS in their research vehicles.

BMW Car IT supports this project, but our main focus is basic research on autonomous driving. For autonomous driving, the challenges in software are to develop suitable architectures for safe fail-operational software systems of complex algorithms. The car must perceive its environment in a robust manner, and handle that the sensors might get noisy signals.It also must allow fail-operational power computing. This means it must execute the functions correctly even if the hardware is faulty, but still, the algorithms are going to require high-performance processors.
Furthermore, this system should be developed cost-efficiently. ROS provides a good framework to join efforts on these research questions.

We are doing research how ROS could be deployed in a series car. This research question is the main topic for the rest of this presentation.

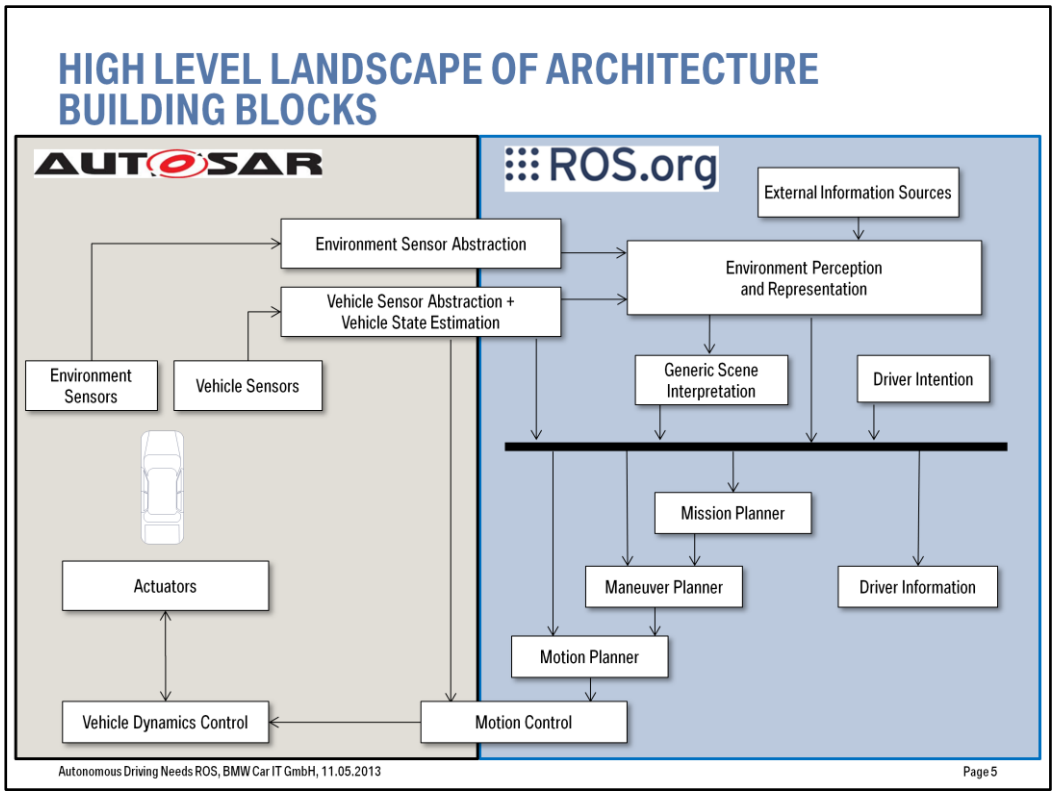


The next three slides discuss the envisioned vehicle network, a high-level logical view on the functional components and the software architecture of our computing platform.

Current vehicles contain two computer domains. The one domain is for the in-vehicle infotainment. For autonomous driving, this system is not in our focus. The other domain is the network of interconnected ECUs, used nowadays for antilock braking, adaptive cruise control and parking assistants. This domain, called the vehicle domain, is built from sensors, controllers, and actuators. The typical characteristics of this domain is that the domain‘s functions must fulfill hard real-time constraints, and the domain has a low change rate, uses established technology and is extremely cost-sensitive because it is installed in every car. However, autonomous driving functions forseeably do not fit into this existing domain.

Autonomous driving needs hardware for two new domains: Hardware for the sensor domain to get a robust environment model and hardware for the connectivity domain to obtain information from other cars and globally stored maps.

The data provided by these two domains are put to use in a central computing platform. This computing platform provides dependable, real-time, power computing. To adjust to the latest technology and evolving algorithms for autonomous driving, the computer platform shall be kept scalable and shall be easily upgraded.



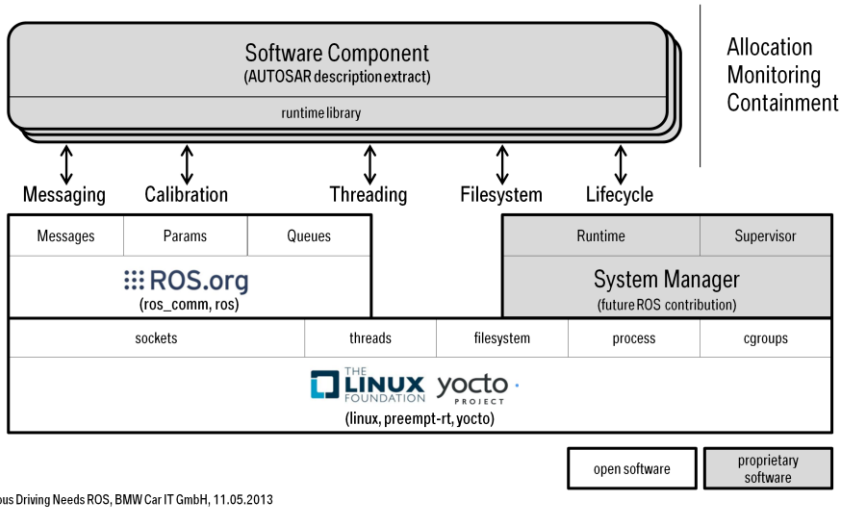
This slide shows the architectural building blocks for autonomous driving and how they could be distributed among the two frameworks, AUTOSAR and ROS. AUTOSAR is the existing automotive standard for model-driven development for embedded systems and defines an operating system running in the vehicle domain. The currently existing driver assistant functions are implemented with the current AUTOSAR technology.

However, for autonomous driving, we envision a more evolved architecture. Advanced functions use input from environment sensors, external information sources, such as maps and car-to-car communication, to build an environment model along the planned driving route. We envision these functions running in ROS nodes on the central computing platform.

This environment model is distributed to a number of planners on different abstraction levels: a motion planner, a maneuver planner, and a mission planer. The motion planner computes the trajectory to a desired location that is a few meters away, draws a virtual line which the car should move along, and controls the motion controller so that it follows that trajectory. The maneuver planner devises the movement on a higher level. It considers actions like changing the lane, overtaking a car, or turning left, and then generates a sequence of points for the motion planner to follow. The mission planer develops the movement on the highest level, and plans which route to take to the final destination, similar to what is currently provided by navigation systems.

SOFTWARE ARCHITECTURE OF THE COMPUTING PLATFORM

- based on proven and widely used technologies
- uses standard Linux features for process isolation
- uses basic ROS features for topic-based communication



Autonomous Driving Needs ROS, BMW Car IT GmbH, 11.05.2013

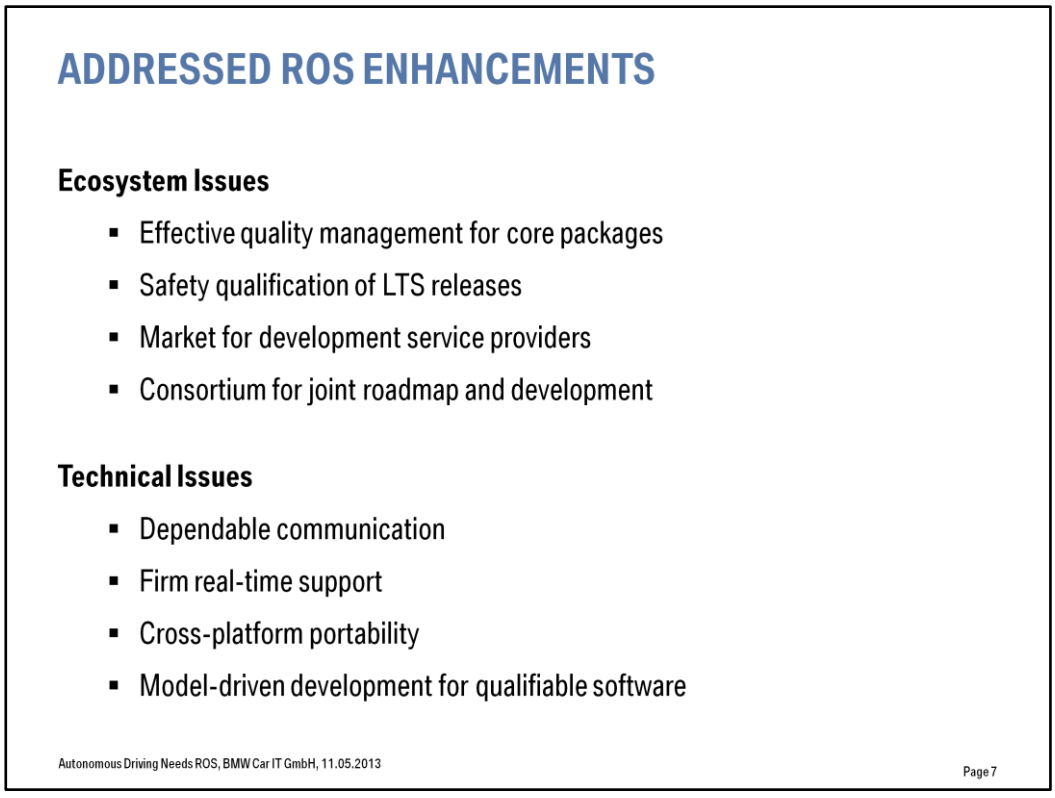
Page 6

On the central computing platform, we envision using these technologies:

The general operating system is provided by a Linux system set up with the Yocto Project Infrastructure. In this system, ROS is used for messaging, calibration and threading, and the application software is running in ROS nodes. To make the transition for developers easy, one can describe the components in AUTOSAR using the existing modeling tools, and generate ROS stubs for those components.

Our open source project meta-ros allows to cross-compile a Linux system with ROS for various different architectures and embedded boards. The System Manager works on system level, provides runtime supervision and monitoring and does a global time-based orchestration. Currently, we are working on the system manager internally, but we are planning to make this open source soon, and contribute it to the ROS community.

Overall, we envision a scalable and dependable platform with an open-source reference implementation that is useful across many different industries, not just for the automotive domain.



The provided ROS contributions that are important to us are:
- a healthy environment, which the ROS community is developing with ROS Industrial and ROS for Products
- quality management and safety qualification
- a market for development services
- a consortium of industrial partners to establish a joint roadmap for development

ROS should support fail-operational execution with dependable communication and firm real-time support.
ROS should be cross-platform portable, e.g., to run ROS on Windows for use of developers and on ARM for use in embedded devices.

For software architects, ROS should allow a model-driven development, as it has been proven to be useful for analyses and assessment of large, complex systems.

FURTHER ROS ENHANCEMENTS

Support for robustness patterns

- Node supervision
- Automatic fail-over to hot or cold standby
- Non-invasive control and data flow monitoring
- Integration of hardware watchdogs

Multi-version support

- Optional elements in message files
- Topic discovery based on message type version

Ideas and Discussions Welcome !

For us of special interest are those issues that we have not seen addressed yet in the community.

These are basically two concerns:
- support for robust execution: In case of failures, ROS needs mechanisms to supervise nodes and allow back-up nodes to automatically take over with a hot or cold standby.
- support for multiple versions: Messages might evolve during the development and exist in different variations.

We suppose that ROS is currently the most suitable existing middleware for the needs of autonomous driving. Most open topics are already addressed by the ROS community, ROS for Products and Industrial ROS. We want to contribute to those communities and push the development by our active engagement. For the specific needs that we have identified, we want to contribute our current work, build up a community to work on these issues together and are looking for partners with common interests.