Apex.OS - A safety-certified software framework based on ROS 2

Jan Becker
My background

Dr. Jan Becker
CEO and Co-Founder Apex.AI
Lecturer at Stanford University

1997-2001 AD with Volkswagen
2002-2006 ADAS at BOSCH
2007-2010 AD at Stanford
2010- Lecturer at Stanford University
2010-2014 Robotics at BOSCH
2011-2015 AD at BOSCH
2016-2017 FF
2009- ROS core development

© 2021 Apex.AI, Inc.
The evolution from ROS 1 to Apex.OS Cert

- **2010**: ROS 1
  - Software framework for robotics
  - Huge adoption in AV industry

- **E2017**: ROS 2
  - Improved code quality
  - Smaller, more optimized code
  - DDS middleware
  - Testing and documentation

- **03/2021**: Apex.OS Cert
  - Functional safety certification (ISO 26262, SEooC, up to ASIL D)
  - Fully deterministic software execution
  - Apex.AI specific extensions

- **Apex.OS**:
  - Real-time execution
  - Real-time data logging
  - Complete documentation
  - Tests, tests, test
  - Support for automotive hardware
  - More tools
  - 24/7 customer support
## The challenge

### IT and Telecommunication Industry

**YESTERDAY**
Hardware-defined phone

**TODAY**
Software-defined smartphone

- **Flexibility**
  - Specific to compute hardware and OS
  - Apps are hardware-agnostic and run on every phone model

- **Scalability**
  - Applications don’t scale
  - Applications scale across the whole ecosystem

- **Labor**
  - Function development is labor-intense
  - Every student can build robust apps

- **Cost**
  - $$$ High and recurring application cost
  - $ Low application cost

### Automotive Industry

**TODAY**
Hardware-defined vehicle

- **Flexibility**
  - Specific to compute hardware and RTOS

- **Scalability**
  - Applications don’t scale across domains

- **Labor**
  - Function development is labor-intense

- **Cost**
  - $$$ High and recurring application cost

Automotive software does not scale to complex software systems required to solve the mega trends autonomous, connected, shared, electric mobility.

OEMs are in need of an end-to-end operating system that is robust and flexible to address all vehicle requirements (ADAS, AD, powertrain, body, chassis, infotainment).¹

Existing prototype software does not scale to automotive production levels of safety.

The solution — SDK-like abstraction by Apex.OS

IT and Telecommunication Industry

YESTERDAY
Hardware-defined phone

TODAY
Software-defined smartphone

Few predefined apps

SDK

OS

Hardware

Millions of apps enabled by SDK

Android/iOS SDK has democratized App development.

Automotive Industry

TODAY
Hardware-defined vehicle

TOMORROW
Software-defined vehicle

Basic functions

Complex functions

SDK

RTOS

Hardware

RTOS

Hardware

SDK-like abstraction for all vehicle domains.
### The benefits

#### IT and Telecommunication Industry

<table>
<thead>
<tr>
<th>YESTERDAY</th>
<th>TODAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware-defined phone</td>
<td>Software-defined smartphone</td>
</tr>
</tbody>
</table>

- **Benefits**
  - **Flexibility**
    - Specific to compute hardware and OS
    - Apps are hardware-agnostic and run on every phone model
  - **Scalability**
    - Applications don’t scale
    - Applications scale across the whole ecosystem
  - **Labor**
    - Function development is labor-intensive
    - Every student can build robust apps
  - **Cost**
    - $$$ High and recurring application cost
    - $ Low application cost

- **Cost**
  - $$$ High and recurring application cost
  - $ Low application cost

#### Automotive Industry

<table>
<thead>
<tr>
<th>TODAY</th>
<th>TOMORROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware-defined vehicle</td>
<td>Software-defined vehicle</td>
</tr>
</tbody>
</table>

- **Benefits**
  - **Flexibility**
    - Specific to compute hardware and RTOS
    - Makes applications independent from hardware and operating systems
  - **Scalability**
    - Applications don’t scale across domains
    - Enables software that scales massively
  - **Labor**
    - Function development is labor-intensive
    - Enables non-expert developers to develop reliable complex applications
  - **Cost**
    - $$$ High and recurring application cost
    - $ Reduces application cost

- **Cost**
  - $$$ High and recurring application cost
  - $ Reduces application cost
The situation — Automotive industry is moving to a centralized hardware architecture

But the required end-to-end operating system doesn’t exist yet¹

All major automotive and robotic players use ROS for prototyping — representing 80% of automotive ecosystem.

ROS provides access to the by far largest developer and user community.

>38,000,000 downloads
>200,000 users
>80,000 software packages
>20,000 developer
>1,000 robots and vehicles

ROS at universities
- >95% of universities use ROS for teaching and research.
- All university competitions such as DARPA / Indy Autonomous Challenge use ROS.

Every robotics student leaving university knows ROS.

ROS is running in
- cars and trucks
- mining and construction
- agriculture
- medical robots
- industrial automation
- personal robots
- drones and eVTOL
- IoT

Validated in many applications

Largest developer and user community
Target

1. A standardized **software architecture with open APIs** to enable mutually compatible solutions ideally across many manufacturers, suppliers, and academia.

2. An **awesome developer experience** to enable developer productivity – based on the understanding that the quality of the developer experience is directly related to their productivity.

3. A **software architecture that scales** to massive software systems.

4. A **software implementation based on modern software engineering practices**.

5. **Abstraction of the complexity** of all underlying hardware and software.

6. **Deterministic, real-time execution, automotive functional safety certification**.

**ROS**
Software architecture considerations

1. hardware abstraction layer
2. OS abstraction layer
3. runtime layer
4. support for various programming languages
5. non-functional performance
6. security
7. safety
8. software updates
9. tools for the development, debugging, recording & replay, visualization, simulation
10. tools for continuous integration and continuous deployment (CI/CD)
11. interfaces to the legacy systems (such as e.g., AUTOSAR Classic)
12. execution management for user applications
13. time synchronization
14. support for hardware acceleration
15. model-based development
ROS architecture

User application

rcl / rclcpp - ROS client library
nodes, services, parameters, timing, console logging, topology graph, utilities

rmw - ROS middleware API
abstraction layer for nodes and services

middleware implementation A
middleware implementation B
middleware implementation C

hardware 1
e.g. x86

hardware 2
e.g. aarm64

hardware 3
GPU

...
Developer experience

Tools, tools, tools
1. Data Visualization
2. System visualization
3. Record and playback
4. Introspection
5. Emulation and simulation
6. Command line tools
7. Development environment
   https://ade-cli.readthedocs.io/
8. Many more at http://wiki.ros.org/Tools
5C’s principle of separation of concerns

1. Functional Entities (Computations) deliver the functional, algorithmic part of a system, that is, the continuous time and space behavior. A Functional Entity can be a composite entity in itself, following the same pattern of composition.

2. A Coordinator to select the discrete behavior of the entities within its own level of composition, that is, to determine which continuous behavior each of the Functional Entities in the composite must have at each moment in time.

3. Functional data Communication handles the data exchange behavior between Functional Entities.

4. A Configurator configures the entities within a level of composition.

5. A Composer constructs a composition by grouping and connecting entities.
Modern software engineering practices

1. An integrated development environment: e.g. centered around Gitlab/Github, CI/CD and docker.

2. An integrated IDE: e.g. Clion. Clion provides all of the state of the art features such as code completion, debugging but also integration of external tools such as e.g. gtest, valgrind, different build tools, doxygen, tool for code test coverage.

3. Deliver often: The steps implementing the development process must be fast to allow agile coding iterations.

4. Test constantly: The local development environment and the CI/CD must be equivalent to be able to reproduce CI failures.

5. Tools follow the purpose (and not the other way around): Integrations with the 3rd party tools, such as a requirements management tool, are tailored to the particular team to allow for the quick fixes and extensions.

6. Single source of truth: The main code repository should be as monolithic as possible and all of the development artifacts (design documents, code, tests, documentation, ...) should be as co-located as possible.
Certification process

ISO 26262, SEooC, part 3, part 6, part 8 processes

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Architecture</th>
<th>Unit Design</th>
<th>V&amp;V</th>
<th>Conf. Reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elicitation, Safety Concept, SW Safety Requirements</td>
<td>UML (unified modeling language), FMEA</td>
<td>SCA (Static Code Analysis), SW practices outline, coverage, FMEA</td>
<td>Req., arch., unit, integration, system, performance, fault injection tests</td>
<td>Safety manual, Restrictions, Traceability</td>
</tr>
</tbody>
</table>

Feature set reduction

Apply real-time and determinism constraints
1. Memory static
2. Remove blocking calls and recursions

Apex.OS Cert

Built-in interfaces
- connext_micro_support
- allocator
- logging
- rclcpp
- threading

Apex_ecu_monitor (native)
Apex_utils (native)
Key steps

1. Making APIs memory static: real-time compliance: Rewrote all non-deterministic runtime memory allocations, blocking calls, and usage of standard STL packages (such as threading).

2. Structural Coverage: 100% statement, branch and MC/DC coverage for all Cert packages as mandated by ISO 26262-6:2018 for ASIL D.

3. FMEA: Extensive safety analysis for every public API to derive additional safety requirements or R&R (restrictions and recommendations) for its users.

4. Requirements traceability:
   1. No formal requirements available from ROS 2 fork.
   2. Wrote several hundred safety and nominal requirements and traced them to codebase and tests using a certified requirement management tool.
Real-time

ROS 2 exhibits the following gaps to enable real-time performance:

- **Non static mem operations**
  - Standard containers
    - Runtime mem allocation
    - Mem fragmentation
  - Standard exceptions
    - Exception throw causes mem allocation
    - Handler lookup non-deterministic due to inheritance
  - Thread priorities, scheduling, pinning
    - No control (std::thread)
  - Standard threading
    - Blocking calls/deadlocks
    - Higher risk of dead locks since no tooling
  - Scheduling based on readiness of data (executor)
    - Increased thrashing

- **non-real-time middleware**

- **Real-time gaps**
Apex.OS addresses the following gaps to achieve real-time performance:

- static mem operations
  - apex::containers
    - apex::string
    - apex::vector
  - apex::map/set
  - apex::malloc
  - Process defined to catch exceptions deterministically

- Apex.Middleware
  - Standard exceptions, but
  - apex::threading
    - Better control over thread priorities, scheduling and pinning

- apex::threading
  - Blocking calls/deadlocks
    - Eliminated
  - Reliance on OS scheduler vs executor
    - Reduced thrashing
Apex.OS is retaining the rich ROS ecosystem
While providing real-time and automotive grade reliability and safety

ROS + = Apex.OS

Largest software development framework for automotive, robotics, autonomous, smart machine applications.

Safety Certification (ISO 26262 ASIL-D)

First and only cross-application SDK certified to the highest level of automotive safety. Certified Apex.OS was honored with CES Innovation Award 2021.
Summary: Enabling software-defined vehicles

1. A standardized **software architecture with open APIs** to enable mutually compatible solutions ideally across many manufacturers, suppliers, and academia.

2. An **awesome developer experience** to enable developer productivity – based on the understanding that the quality of the developer experience is directly related to their productivity.

3. A **software architecture that scales** to massive software systems.

4. A **software implementation based on modern software engineering practices**.

5. **Abstraction of the complexity** of all underlying hardware and software.

6. **Deterministic, real-time execution, automotive functional safety certification**.
Outlook

Components:
- Eclipse Cyclone DDS
- Eclipse iceoryx
- SOME/IP

Integrated in Apex.Middleware:
- Integrated with Apex.OS Cert
- Interoperable with AUTOSAR Adaptive (ara::com and SOME/IP)
- DDS Security
- Automotive grade and supported

Scope
- Proof-of-Concept (PoC)
- Pre-production
- Production

Apex.Middleware Cert:
- Includes developer tools
- Professionally supported
- ISO 26262 certification
Apex.OS - A safety-certified software framework based on ROS 2