Solution Adaptive MICROSAR

Ready for Next Generation ECUs
Automotive Trends

Cloud / Backend
- support of high performance processors
- high bandwidth
- service based architectures
- open source, agile development
- dynamic and updatable
- internet

Embedded Systems
- safe
- secure
- embedded integration and debugging
- automotive supply chain
- automotive communication protocols
- automotive diagnostics

AUTOSAR Classic
Automotive Trends

Adaptive – best of two worlds

Cloud / Backend
- system and mobility strategies
- deep learning

On board Supercomputers
- multipurpose computing servers
- connectivity, gateways, HMI
- automated driving
  - mastered by OEM

Embedded Systems
- intelligent sensors and actuators
- basic functions
- fallback computing

AUTOSAR Adaptive

Adaptive AUTOSAR
- high bandwidth
- service based architectures
- open source, agile development
- dynamic and updatable
- internet

safe
secure
embedded integration and debugging
automotive supply chain
automotive communication protocols
automotive diagnostics
automotive supply chain
automotive communication protocols
automotive diagnostics
Introduction

Being Prepared for the Next-Generation of ECUs

Adaptive MICROsAR is a complete basic software solution up to ASIL D

Seamless interoperability with classic AUTOSAR ECUs

Additional, high performance ECUs hosting applications for upcoming use cases

Applications installed and started during runtime

Development of applications in the ecosystem of POSIX-based OS (Linux, PikeOS, QNX, Integrity, ...)

Infotainment

ADAS

Connectivity

Dynamic Software Platform
Introduction

Statically connected HW Resources

- Hardwired video lines between ECUs
- Pre-defined CAN messages on bus
- Exclusive camera usage
Introduction

Flexible use of HW Resources

- Smart sensors/actuators provide HW over service interface
- All ECUs connected via Ethernet
- Compound service, using base services as lower layer
- Applications can provide services for e.g. HMI integration
- No function oriented wiring
Drivers for Adaptive AUTOSAR

**Introduction**

**Infotainment**
- 2D/3D acceleration support in POSIX systems
- Video Codecs, Streaming support, multi-media library, etc. ...

**Connectivity**
- Car-2-X (LTE, Wi-Fi, GPS, etc.)
- Multimedia (USB, SD-Card, NFC, etc.)

**Highly Automated Driving**
- Image- and preprocessing of Camera/Radar/LIDAR
- Sensor Fusion and Machine Learning

**Dynamic Software Platform**
- “App-Store” for automotive applications
- Installation and update over the air
Introduction

Use Cases for POSIX/Virtualization in Automotive Systems

1. POSIX besides MICROsAR (previously used set-up)
2. Applications as Driver
3. POSIX besides MICROsAR (current safety set-up)
4. Adaptive Autosar (upcoming perspective)
Agenda

Automotive Trends
Introduction

▶ Fundamentals
Details and Functional Clusters
Activities and Roadmap
## Fundamentals

### AUTOSAR Product Comparison

<table>
<thead>
<tr>
<th>AUTOSAR Classic Platform - CP</th>
<th>AUTOSAR Adaptive Platform - AP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Layer</strong></td>
<td><strong>SWC</strong></td>
</tr>
<tr>
<td>Application Software Component</td>
<td>OS</td>
</tr>
<tr>
<td>Actuator Software Component</td>
<td>COM</td>
</tr>
<tr>
<td>Sensor Software Component</td>
<td>API</td>
</tr>
<tr>
<td>AUTOSAR Interface</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>AUTOSAR Interface</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td><strong>Runtime Environment</strong></td>
<td><strong>SWC</strong></td>
</tr>
<tr>
<td>System Services</td>
<td>OS</td>
</tr>
<tr>
<td>Memory Services</td>
<td>COM</td>
</tr>
<tr>
<td>Communication Services</td>
<td>API</td>
</tr>
<tr>
<td>I/O Hardware Abstraction</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Onboard Device Abstraction</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Microcontroller Drivers</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Memory Hardware Abstraction</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Communication Hardware Abstraction</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>I/O Drivers</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td><strong>Microcontroller</strong></td>
<td><strong>AUTOSAR Runtime Environment for Adaptive Applications</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SWC</strong></td>
</tr>
<tr>
<td><strong>Adaptive Platform Services</strong></td>
<td>OS</td>
</tr>
<tr>
<td>Service (nm)</td>
<td>COM</td>
</tr>
<tr>
<td>Network Management</td>
<td>API</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Service (sl)</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>State Management</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Signal to Service Mapping</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td>Update &amp; Configuration Management</td>
<td>AUTOSAR Interface</td>
</tr>
<tr>
<td><strong>Adaptive Platform Foundation</strong></td>
<td><strong>(Virtual) Machine / Hardware</strong></td>
</tr>
</tbody>
</table>

### All modules completely specified
- Developed in C
- Whole stack compiled and linked in one piece
- Will still remain in the current focus
- Configuration compiled in

### Less modules, only API specification
- Developed in C++
- Services as POSIX processes, separately installable
- Service oriented communication (SOME/IP)
- Configuration loaded from manifest files
AUTOSAR Product Comparison

**AUTOSAR Classic Platform - CP**

- Application Layer
- System Services
- Onboard Device Abstraction
- Microcontroller Drivers
- Memory Services
- Memory Hardware Abstraction
- Communication Services
- Communication Hardware Abstraction
- I/O Hardware Abstraction
- Complex Drivers
- I/O Drivers

**AUTOSAR Adaptive Platform - AP**

- SWC
- ARA
- SWC
- ARA
- SWC
- ARA

**Runtime Environment**

- API (sync)
- API (exec)
- API (com)
- Network Management
- Service (nm)
- Service (diag)
- Adaptive Platform Services

- Time Synchronization
- Execution Management
- Communication Management
- Management
- Diagnostics
- Service (nm)
- Service (diag)

- API (phm)
- API (log)
- API (per)
- Service (bin)
- Service (bin)

- Platform Health Management
- Logging & Tracing
- Persistency
- Signal to Service Mapping
- Service (bin)
- Service (bin)

- API (iam)
- API (rest)
- API (crypto)
- State Management
- Service (bin)
- Service (bin)

- Identity Access Management
- RESTful
- Cryptography

- State Management
- RESTful
- Cryptography

- API (core)
- POSIX PSE51 / C++ STL
- Operating System
- Adaptive Platform Foundation

- Core Types
- Operating System
- Adaptive Platform Foundation

**Real Time Requirements**

- Safety Critical

- Computing Power
Adaptive Applications

Manifest

Instance Configuration

Application (1:n Executables)
POSIX Process

INIT:

RUN:
Thread  Thread  Thread

SHUTDOWN:

PSE51
C++ Stdlib

ara::com

Direct API

Adaptive AUTOSAR Services

Adaptive AUTOSAR Foundation

Application

> Multi-threaded
> Execution states
> Manifest contains platform related information (recovery action, dependencies to services or libraries)
> Instance configuration contains application specific static information (variant, options, ...)

Interfaces

> ara::com for communication with adaptive services (basic services and user applications)
> PSE51 is the usable OS API subset
> The Adaptive AUTOSAR Foundation clusters (Execution Management, Persistency, etc.) are available via direct APIs
Fundamentals

Vector’s Adaptive Implementation

- Implementation available for several platforms
- Application logic is strictly separated from configuration (see /opt/ deployment directory)
- Code examples based on Vector’s implementation
- Guided configuration via IDE in Eclipse Plugin
Fundamentals

Tools and Workflow

**Application Code**
- Logic
- libara
- libsoméip

**Service Description (ARXML)**
- AppSWCTypes
- Port
- ServiceInterface
- SOME/IP Config

**Deployment Package**
- BIN
  - ./bin/myApp
- Execution Manifest
  - ./etc/MANIFEST.arxml
- Instance Manifest(s)
  - ./etc/instance1.arxml
  - ./etc/instance2.arxml

**Vehicle**
- Installed APP
  - BIN
- Executable Config. (JSON)
- Instance Config. (JSON)
- Execution Management
- Diagnostics
  - ComServer
  - POSIX IPC
  - SOME/IPd
  - BSD Sock

**Generators**
- Port
- Port
- Port
- Port
- Port

**Compiler**
- POSIX IPC
- SOME/IP Serializer
- E2E Serializer
- Proxies / Skeletons

**Authoring Tool**
- AppSWCTypes
- Port
- ServiceInterface
- SOME/IP Config
Fundamentals

Tooling: DaVinci Adaptive Tool Suite

1. Assistants for various tasks like creation of SOME/IP deployment
2. Easy to understand DSL to represent ARXML models. With linting support
3. Auto-completion for references and model elements
4. Built-in CFG-5 generators. Direct modelling feedback and resolution suggestions
5. Cheat Sheets guide through the process of service creation
PREEvision: Adaptive system design with PREEvision

1. Diagram-based design of service interfaces
2. Modeling of implementation details of the service interfaces
3. Design of the SW components, executables and adaptive applications
4. Table-based editing of deployment aspects as SOMEIP IDs
5. Graphical design of Ethernet topologies
6. Assistant for creation and mapping of service instances on machines
7. Import and export of different model subsets in AUTOSAR XML
Adaptive MICROSAR Evaluation Bundle

- Test your application directly in native environment
- Implement your services using Eclipse
- Adaptive MICROSAR source included
- Prepared build scripts for native Linux

Bundle is available off-the-shelf and includes:

- Free recorded training Webinar
- Application Developer Guide
- DaVinci Adaptive Tool Suite (1 year license)
Agenda

Automotive Trends
Introduction
Fundamentals

- Details and Functional Clusters
Activities and Roadmap
Details and Functional Clusters

Communication: ara::com

- Service-oriented communication

- **Location-transparent**

- Supports multiple communication bindings
  - AUTOSAR model defines available bindings for each service provider and consumer
  - Explicit support for optimized shared memory implementations

- Applications **connected at runtime** (Service Discovery)
  - Find service instances dynamically without hardwiring in model
  - Connection between proxies and skeletons can be recovered

- **Real-time support**: Developers’ choice of polling or event-driven processing of communication
Communication: Method Calls

- Communication initiated by the service consumer
- Bidirectional data flow
- N:1 communication: method can be called by multiple consumers
- Provider controls how parallel method calls are handled (serial, full parallel)

1. Call method as you would call a function:
   \[ f = \text{Proxy.method}(\text{arg}, \ldots) \]
   "f" is the handle for the call

2. Method call transmitted

3. Method implementation called: ReturnType
   
   ```cpp
   \text{Skeleton::Method(\text{arg},\ldots)} \{
   \quad \text{return return_value;}
   \}
   ```

4. Call result transmitted

5. Call result can be obtained using \( f.\text{get}() \)
Communication: Sending Events

- Communication initiated by the service provider
- Unidirectional data flow from provider to consumers
  - 1:n communication
- Consumer controls buffering strategy of events
- Event has a value only in the instant that it occurs

```
skeleton.event.send(value)
```

(1) Event stored in “invisible” buffer
(2) Event containing value
(3) User calls event.update() – Predefined number of events moved to visible buffer
Execution Management: ara::exec

- OS launches Execution Manager (EM) (PID1, "init")
- EM inspects system for installed applications
  - E.g., scan filesystem in /opt/ for application manifests
- EM runs startup applications (fork(), exec())
  - e.g., bring up IP stack

- EM consults Machine State Manager to determine desired machine state
  - Machine state defines set of applications desired to run
- EM starts/stops applications to reach desired machine state (fork(), exec(), signal(SIGTERM))
  - EM configures scheduling parameters & resource limits
  - Configuration data obtained from application manifest
- EM monitors for machine state changes or process termination
Details and Functional Clusters

Execution Management: Connecting Applications - Example

- Communication via API (library with IPC included)
- Functionality provided by Execution Manager
  - API for applications to report application state (e.g. kInitializing, kRunning, kShuttingdown)
  - API for Machine State Manager to
    - Register as MSM
    - Request machine state
    - Get current machine state

Adaptive Application - AA

Machine State Manager

ApplicationState

ReportApplicationState()

MachineState

SetMachineState()

GetMachineState()

Execution Management - exec
Diagnostics: Overview

No fundamental changes to existing diagnostic workflows (like development, production, workshop,...) due to Adaptive Platform

- Main Tasks
  - ISO 14229-5 (UDSonIP)
  - Including fault-memory (DTC) handling
  - Including transport layers (i.e. DoIP – ISO 13400-x)

- Configurable via AUTOSAR Diagnostic Extract (DEXT)

- ARA service
  - Uses ara::com interfaces
Diagnostics: Configuration workflow (DEXT)

Details and Functional Clusters

Diagnostic Design

System Design

CANdela Studio

DEXT

PREEvision

MANIFEST (Design)
Details and Functional Clusters

Persistency: ara::per

- library based access to non-volatile memory for Adaptive Applications.

- **Key-Value Storage**
  - Multiple values stored in one storage location
  - Addressing of single values by using a key as identifier
  - Multiple storage locations/databases can be used
  - Database format not specified by AUTOSAR

- **Stream Storage**
  - Raw access to storage locations/files
  - Used for access to files in any format
  - API derived from C++ Standard Library `std::fstream` classes

Adaptive Application - AA

KeyValueDatabaseInterface

FileProxyInterface

Key-Value-Storage
- Create/Recover/Reset
- GetValue
- SetValue

File-Proxy
- Create/Recover/Reset
- CreateAccess
- DeleteKey
- Read / Getline

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Identity and Access Management

- Offers intra ECU and inter ECU access management
- A policy decision point decides whether resource access is granted or not

1. Request action
2. Is Application “x” authorized?
3. Yes
4. Perform request (e.g., access resource, communicate, get information, etc.)
Agenda

Automotive Trends
Introduction
Fundamentals
Details and Functional Clusters

Activities and Roadmap
Activities and Roadmap

Safety as a necessity for high performance ECUs

- Use cases such as highly automated driving easily demand safety up to **ASIL D**

- Architecture in high performance controllers is complex due to many involved abstraction layers:
  - BSP from semiconductor vendor
  - Hypervisor
  - Multiple Operating Systems
  - POSIX libraries
  - Adaptive AUTOSAR Basic Software
  - Service Oriented Applications Components

- Typically many vendors involved for these components

- **Vector is providing the complete and ready to go solution for such controllers out of one hand.**
Vectors Adaptive Activities

- Series production projects have been started for many customers
- Linux
- QNX
- PikeOS
- Integrated tool concept
- Evaluation Bundle
- Products synchronized with Autosar specification
- Multiple POSIX systems integrated
- All feature teams are covered
- Active participation in all working groups
- Series Production Development
- Specification of Adaptive platform
- Available Off-the-shelf Products
# Planned Adaptive MICROsAR Roadmap

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Adaptive MICROsAR** development started in 2015
- **Adaptive MICROsAR** used in many evaluation and prototyping projects
- **Adaptive MICROsAR** used in the first series production projects
For more information about Vector and our products please visit

www.vector.com

Author:
Dr. Markus Oertel, Mirko Tischer
Vector Germany