Adaptive AUTOSAR

Ready for Next Generation ECUs
Introduction

Being Prepared for the Next-Generation of ECUs

Additional, high performance ECUs hosting applications for upcoming use cases

With **Adaptive MICRO SAR**, Vector is providing a complete basic software solution

Seamless interoperability with classic AUTOSAR ECUs

Applications installed and started during runtime

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

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

Infotainment

ADAS

Connectivity

Dynamic Software Platform
Introduction

Drivers for Adaptive AUTOSAR

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

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

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

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

Use Cases for POSIX/Virtualization in Automotive Systems

- **Consolidation of existing ECUs**
  - Core 1 (MSR 1)
  - Core 2 (MSR 2)
  - IPC
  - MEM
  - Mini-Hypervisor

- **Applications as Driver**
  - Process 1 (nat. POSIX)
  - Process 2 (MSR)
  - IPC
  - MEM
  - POSIX

- **POSIX besides MSR**
  - (current set-up)
  - (current QM set-up)

- **Adaptive Autosar**
  - (upcoming perspective)

- **POSIX besides MSR**
  - (previously used set-up)
  - (current safety set-up)
Agenda

Introduction

- **Fundamentals**
  - ara::com
  - ara::em
  - ara::diag
  - ara::per

Roadmap
### AUTOSAR Product Comparison

#### AUTOSAR Classic Platform - CP
- 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

#### AUTOSAR Adaptive Platform - AP
- 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**
  - Application Software Component
  - Actuator Software Component

- **Runtime Environment**
  - Memory Services
  - Onboard Device Abstraction
  - Microcontroller Drivers

- **Complex Drivers**
  - I/O Hardware Abstraction
  - Communication Services

- **Microcontroller**

**AUTOSAR Adaptive Platform - AP**

- **SWC**
  - ARA

- **Runtime Environment for Adaptive Applications**
  - ARA

- **Adaptive AUTOSAR Services**
  - Time Management
  - Execution Management
  - Software Configuration Management

- **API**
  - Operating System
  - Persistence

- **Bootloader**

**Real Time Requirements**

**Safety Critical**

**Computing Power**
Adaptive Architecture

Applications

App1
POSIX Process
ara::com ara::em ara::pers

App2
POSIX Process
ara::com ara::em ara::pers

Middleware
ara::com

SOMEIPd
Service Discovery

Platform
ara::com ara::em ara::pers

SCM Service

ara::com ara::em ara::pers
Diagnostic Service

ara::em
EM (Execution Manager)

Persistency

BSD Socket

BSD Socket for DoIP

POSIX OS
Adaptive Applications

- **Application**
  - Multi-threaded
  - Execution states
  - Manifest contains platform related information (recovery action, dependencies to services or libraries)
  - Instance config 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

Adaptive Implementation

- VMs with prototype implementation (Based on YOCTO)
- Currently supported platforms
  - Renesas R.Car H3 (Arm)
  - ST Telemaco3P (Arm)
  - Minnowboard (Intel)
  - More in preparation

- Set of example services based on code in AUTOSAR GIT
- Additional services in development
- Tool supported service specification
Adaptive MICROSAR Development Environment

- Implement your services using preconfigured Eclipse
- Preconfigured Yocto build environment
- Adaptive MICROSAR source included
- Test your ECU directly in QEMU

Furthermore included:

- 2 Day training at Vector for 1 Person
- Application Developer Guide
- Tooling for creating service interfaces
- Generators for proxies and skeletons
**Fundamentals**

**Tools and Workflow**

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

- **Application Code**
  - Logic
  - libara
  - Port
  - Proxies / Skeletons
  - SOME/IP Serializer
  - E2E Serializer
  - POSIX IPC

- **Deploy Package**
  - /opt/myApp/
    - BIN
      - ./bin/myApp
    - Application Manifest
      - ./etc/MANIFEST.arxml
    - Instance Manifest(s)
      - ./etc/instance1.arxml
      - ./etc/instance2.arxml

- **Vehicle**
  - Installed APP
    - BIN
  - Application Manifest (JSON)
  - Instance Manifest(s) (JSON)
  - EM
  - ComServer
  - Diag
    - POSIX IPC
    - SOME/IPd
    - BSD Sock

- **Authoring Tool**
  - Generators
    - ServiceInterface
    - SOME/IP Config

- **Compiler**

- **Software Configuration Management**

**Generators**
- Static
- Generated

**Compiled**

- myApp
  - Application Manifest
    - ./etc/MANIFEST.arxml
  - Instance Manifest(s)
    - ./etc/instance1.arxml
    - ./etc/instance2.arxml

**Deployed**
- /opt/myApp/
  - BIN
    - ./bin/myApp
Agenda

Introduction
Fundamentals

- ara::com
- ara::em
- ara::diag
- ara::per
Roadmap
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:
   \[ f = \text{Proxy}.\text{method}(arg, \ldots) \]
   "f" is the handle for the call

2. Method call transmitted

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

4. Call result transmitted

5. Call result can be obtained using \[ f.\text{get}() \]
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
Agenda

- Introduction
- Fundamentals
  - ara::com
  - **ara::em**
  - ara::diag
  - ara::per
- Roadmap
Execution Management Overview

- 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
Connecting the ExecutionManager to Applications - Example

- Communication via API (library with IPC included)
- Functionality provided by EM
  - 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
Agenda

Introduction
Fundamentals
ara::com
ara::em
➤ ara::diag
ara::per
Roadmap
Diagnostic Manager - Overview

No fundamental changes to exiting 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

---

**Application (Software Components)**

- Diagnostic Coding
- Diagnostic Measurements
- Diagnostic Routines
- Diagnostic Monitor

**Diagnostic Manager**

- Diagnostic Request
- Diagnostic Response

**Tester**
Configuration workflow (DEXT)
Introduction
Fundamentals
ara::com
ara::em
ara::diag
► ara::per
Roadmap
In AUTOSAR Adaptive the Persistency cluster provides a library based implementation to access non-volatile memory to Adaptive Applications so that data can be stored non-volatilely.

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
Agenda

Introduction
Fundamentals
ara::com
ara::em
ara::diag
ara::per

Roadmap
Planned Adaptive Roadmap

**ASR Release 17-03:**
- Execution Management
- Communication/Middleware
- DLT
- Diagnostics
- Persistency

**ASR Release 17-10:**
- Platform Health Management
- RESTful Services
- Signal based communication
- Security Features
- Time Synchronization
- E2E Protection (Draft)
- Package Management (Draft)

**ASR Release 18-03:**
- Container Support
- Package Management
- E2E Protection
- SW-Lockstep
- FR + LIN Bus
- Crypto Hardware
- Network Management

**ASR Release 18-10:**
- TBD

2017

- Deliveries based on Adaptive AUTOSAR Code
- Vector Adaptive modules replace community source parts gradually as development is progressing

2018

- Adaptive MICROSAR R1: Development Release

2019

- Adaptive MICROSAR R2: Production Release (QM)

2020

- Adaptive MICROSAR R4: Production Release (Safe)
For more information about Vector and our products please visit

www.vector.com

Author:
Dr. Markus Oertel
Vector Germany