Service Oriented Architecture: Design and Implementation Using Automotive Linux BSP

Marius Rotaru
Software Architect & Technical Director

Catalin Udma
Linux Software Architect

June 2019 | Session #AMF-AUT-T3657
Agenda

• Introduction to Service Oriented Architecture Frameworks
• NXP’s infrastructure for SoA
• Applications & Use Cases
Introduction to Service Oriented Architecture Frameworks
Vehicle Architectures
Mega Trends: Safe and Secure Mobility

**Autonomy**
- Different sensor types
- Data fusion: Safe Processing with Integrated AI capabilities
- Fail operation
- Big Data

**Electrification**
- Power Efficiency
- Battery Management
- Electrification Levels
  - Hybrid, full electric...
- Broad range of solutions
- Need for standardization

**Connectivity**
- V2X, 5G, Digital Radio
- Diagnostics / Prognostic Health Management
- OTA Update Management
- Analytics (edge to cloud)
- Software-centric solutions
- System security

Major Changes in Network Topology and E2E Architectures
Vehicle Architectures
Different networking models across the 4 options
Vehicle Architectures
Mega Trends: Embedded Software become Software

Technology Trends
AUTONOMY
ELECTRIFICATION
CONNECTIVITY

E/E Implication

- ECU Platform
- Topology
- Communication
- OSEK/VDX
- Signal Comm
- Static configuration
- Rich Operating Systems (e.g. Linux)
- Service Oriented Architecture
- Dynamic configuration
Signal vs Service Oriented Communication Paradigms

Background

With signal-oriented data transmission information is sent when the sender sees a need, such as when values are updated or changed, independent of whether these data are currently needed by a receiver in the network.

Signal-oriented data transmission is used on classic bus systems (CAN, LIN, FlexRay).

Service-oriented data transmission, a sender only sends data when at least one receiver in the network needs this data. The advantage of this procedure is that the network and all connected nodes are not loaded by unnecessary data.

Service-oriented data transmission is mainly using Ethernet bus.

SOA Concepts:
- Service Consumer
- Service Provider
- Service Broker
- Service Bus
Service-oriented Architecture – SoA
All about Middleware

Source: AUTOSAR_GuidedTour.ppt

Source: ISITA_World_Summit_2015_FUERST_Simon__for_web__.pdf

Source: AUTOSAR_AdaptivePlatformFor_EXP_TechnicalOverview.pptx
Service-oriented Architecture (SoA)

Service-orientated Architecture (SoA) is a way of designing software where the participating components provide and consume services over a predefined protocol over a network.

- **Functionality Separation:** Units, Services
- **Functionality Distribution:** Ownership domains
- **Interoperability:** Interfaces, Protocols
- **Reusability:** Combination, Integration

A **service** is a discrete unit of functionality which can be remotely accessed and independently updated.
Service-oriented Architecture (SoA)

Asynchronous Remote Procedure Calls

Implementation of RPC for SoA
Service Oriented Middleware – SOME/IP

SOME/IP = Scalable service-Oriented Middleware over IP

➢ SOME/IP provides service oriented communication over a network

➢ SOME/IP supports a wide range of middleware features:
  • Serialization
  • Remote Procedure Call (RPC)
  • Service Discovery (SD)
  • Publish/Subscribe (Pub/Sub)
  • Segmentation of UDP messages

➢ SOME/IP can be implemented on different operating systems

➢ SOME/IP is used for inter-ECU Client/Server Serialization

➢ SOME/IP allows applications to communicate.

Source: SOME-IPIntro.pdf
Service Oriented Middleware – SOME/IP Services

- **Request/Response** – a method call with Request and Response messages

- **Fire&Forget** – a method invocation with just a Request message (does not support answers and errors)

- **Event** – a Fire&Forget callback, that is sent out by the Server (e.g. cyclically or on change)
  - Sent from Server to Client (Similar to regular CAN messages)

- **Field** – represents a remote accessible property that includes Getter/Setter and/or Notification (similar to a property on MOST)
**Service Oriented Middleware – SOME/IP**

**Service Discovery - SD**

**SOME/IP-SD** is used to:
- Locate service instances.
- Detect if service instances are running.
- Implement the **Publish/Subscribe** handling

![Image source: realtimeapi.io/hub/publishsubscribe-pattern/](realtimeapi.io/hub/publishsubscribe-pattern/)
Service Oriented Middleware – SOME/IP
Pros and Cons\[1\]

Main Advantages:
- Coexistence with existing system  
  > No functional Loss
- High Data Rate and Unicast  
  > Increase data transfer amount
- Low Transportation Overhead
- Dynamic IP Addressing  
  > Gain in maintainability and flexibility

Possible Issues:
- Computational Overhead due to complex architectures
- Increase Storage Requirements
- Single Point of Failure (e.g. Switch Malfunction)

Recommended usage:
- Suitable for driver assistance and Infotainment systems
- Still too complex for Hard Real-time system (e.g. motor control)

Several Open Source implementations (e.g. GENIVI vsomeip)

Service Oriented Middleware – DDS
DDS = Data Distribution Software

Standard-based Integration Infrastructure for Critical Applications

Family of specifications

Images source:
http://www.slideshare.net/SumantTambe/communication-patterns-using-datacentric-publishsubscribe
Service Oriented Middleware – DDS
DDS Standard

• DDS is the Proven Data Connectivity Standard for the IoT
• OMG: world’s largest systems software standards org
  – UML, DDS, Industrial Internet Consortium
• DDS: open and cross-vendor
  – Open Standard and Open Source
  – 12+ implementations

DDS Wire Protocol (RTPS)
• Peer to peer
• Transport-independent QoS-aware and Reliable Communication
  – Including multicast, for 1-many reliable communication
• Any data size over any transport.
• Automatic Discovery and Presence Plug and Play
• Decoupled
  – Start applications in any order
• Support for Redundancy
  – Multiple data sources
  – Multiple network paths
• High performance native “wire” speeds

Images source:

OMG: Object Management Group
RTPS: Real-Time Publish/Subscribe
Service Oriented Middleware – DDS
Communication pattern based on Data-centric Publish/Subscribe

Provides a “Global Data Space” that is accessible to all interested applications.
- Data objects addressed by Domain, Topic and Key
- Subscriptions are decoupled from Publications
- Contracts established by means of QoS
- Automatic discovery and configuration

Images source: http://www.slideshare.net/SumantTambe/communication-patterns-using-datacentric-publishsubscribe
Service Oriented Middleware
DDS as Core Connectivity Framework

Images source: http://www.slideshare.net/SumantTambe/communication-patterns-using-datacentric-publishsubscribe
Adaptive AUTOSAR
as Service Oriented Communication Framework

- *ara::com* is the Communication Management API for the AUTOSAR Adaptive Platform.
- Aims to be communication framework independent
- Was initially built around SOME/IP and follows most of its principles
- Based on a proxy/skeleton SOA architecture
- Especially tailored for Modern C++ (C++11 in External APIs, C++14 in Internal APIs)

Source: AUTOSAR_AdaptivePlatformFor_EXP_TechnicalOverview
Service Oriented Middleware – Adaptive AUTOSAR
ARA::COM - Service-oriented Communication – Proxy/Skeleton Paradigm

Two code artifacts are generated from AUTOSAR ARXML service description.

- **Service Proxy: facade**: an instance of a generated class, which provides methods for all functionalities the service provides.
- **Service Skeleton: instance** of a generated class which allows to connect the service implementation to the Communication Management transport layer.

- Bindings can be implemented for REST, DDS or other Middleware Transport Layers that support publish / subscribe / event patterns
- SOMEIP is the default transport layer available on the shelf for ARA::COM
- Transport Layer is not necessary network, it can be shared memory or direct function calls if client and service are running the same ECU / address space

Source: AUTOSAR_AdaptivePlatformFor_EXP_TechnicalOverview
AUTOSAR Adaptive – Architectural Overview

ARA::COM – Language and Network Binding

- An Adaptive Application may use different communication bindings underneath the ara::com common API.
- DDS is placed parallel to other network bindings such as SOME/IP.

Not standardized
- analog to Com_SendSignal()
Service Oriented Middleware – ROS

(A) ROS Community: ROS Distributions, Repositories

(B) Computation Graph: Peer-to-Peer Network of ROS nodes

(C) File-system level: ROS Tools for managing source code, build instructions, and message definitions.
SW Environment: SoA

Definition

- **SoA**: Service Oriented Architecture
  - Applications built with ‘service’ layer of abstraction
  - SoA App is not bound to specific OS, SOC, or even ECU
  - SoA Apps also referred to as ‘Services’
  - Framework that will transform how vehicle features are built and deployed – portability at forefront (static or dynamic)

- SoA Framework examples:
  - Adaptive AUTOSAR
  - MQTT
  - ROS

Why

- **Motivation**
  - Ease of development, deployment & integration
    - OEMs move away from deploy features by ECU, to features by SoA Apps.
    - Tier1 moves to deliver SoA Apps to OEMs, rather than ECUs.
    - Simplifies the OTA deployment

- **Challenges for SOC Arch**
  - Difficult to HW enforce isolation of SoA Apps for security / safety
  - **SoA Framework dictates performance, security & safety. Framework provider & SOC provider need to work closely to make use of hardware features.**
NXP’s Infrastructure for SoA
Automotive Linux BSP
Overview of Automotive Linux BSP

- A Linux BSP for all NXP Automotive Platforms
- Targeting ADAS, C&S and Disty Market
- Integrated with SDKs (Vision, Radar, Ethernet)
- Quarterly Releases
- A single package for multiple SOCs
We are building an Infrastructure leveraging huge open source ecosystem and various communities targeted automotive software.
Automotive Linux BSP - Product Architecture

**Middleware**
- Services
  - Communication
  - Storage
  - Bootloaders
  - UEFI

**Applications**
- Ubuntu
  - Graphics
  - DCU
  - Framebuffer
  - GPU
  - VIU
  - Encoders

**Tools**
- Customer Applications
- NXP GCC

**Samples**
- benchmarks
- Documentation

**Kernel**
- Provided by SDKs
  - SoC1
  - SoC2
  - SoC3
  - SoC4
  - BlueBox
  - S32X...

**IPC/F**
- Hypervisor (XEN)
- Hypervisor (XEN)
Adaptive AUTOSAR – Reference Implementation
Adaptive AUTOSAR ready to use/build ecosystem

Adaptive Application

Adaptive AUTOSAR Foundation

OS Abstraction Layer (OSAL)

Linux | QNX | Integrity

OS (+ BSP)

Linux BSP | QNX BSP

NXP Develop
NXP and/or 3rd party
3rd party
Inter-Platform Communication Framework (IPCF)

- Multiple homogeneous or heterogeneous processing cores
- Located on a single chip or on multiple chips in a circuit board
- Running multiple OSes
- Communicating over various interfaces:
  - Ethernet
  - PCIe
  - USB
  - UART, SPI
  - Shared memory
IPCF

API

MCAPI

IPCF lib

CORE IPCF

OS Abstraction Layer

SoC Abstraction Layer

Transport Abstraction Layer

Drivers

VETH / VTTY

UART

USB

Shared mem

PCle

ETH

Nodes

Endpoints

ISO26262
IPCF – Linux2AUTOSAR over Shared Memory

**Linux Userspace**: vSomeIP, OpenSSL, IPsec

**Linux Kernel**: TCP/IP stack

**Customer Module**: vSomeIP, OpenSSL, IPsec

**Virtual ETH Device**: TCP/IP stack

**Zero-copy API**: Customer module

**SHM Driver**: Customer Apps

**IPCF Lib**: MCAPI

**POSIX Socket API**: MCAPI

**Customer Apps (SW-C)**: SomeIP, SSL/TLS, SecOC

**IPCF CDD**: MCAPI

**Virtual ETH CDD**: MCAPI

**Customer CDD**: Com Stack [PduR/Tcplp]
Virtual Ethernet

Standard Ethernet Linux interface

Implementation:
- Virtual Ethernet Network Device
- PCIe driver

BlueBox

BlueBox

PCle Communication

$ ifconfig pcie0 192.168.1.1

$ ping 192.168.1.2
PING 192.168.1.2 56(84) bytes of data.
64 bytes from 192.168.1.2: time=0.34 ms
64 bytes from 192.168.1.2: time=0.35 ms

$ ifconfig pcie0 192.168.1.2

Interfaces in LS Linux
- pcie0

Interfaces in S32V Linux
- pcie0

Virtual Eth Instances
- veth

PCIe Driver (RC)
- S32V

PCIe Driver (EP)
- RC

LS2
S32V

COMPANY PUBLIC | 31
Linux BSP Delivery: Easy of Use Using Yocto

Yocto Project: An open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products.

- Yocto Upstream
  - Poky
  - Open embedded
  - Virtualization

- Flexera & CAF
  - Documentation
  - Yocto Binaries
  - Quality package

- CAF: Open-Source Code

- NXP source code
  - Linux Kernel
  - U-boot
  - Kernel modules (eth, PCIe)
  - Linux Userspace

- NXP Yocto BSP
  - Yocto Upstream Layers
  - Yocto BSP NXP Specific layers

- Testing and Validation
  - Yocto testing

- NXP Delivery: Flexera & CAF
  - CodeAurora FORUM
Ubuntu Delivery – The same delivery mechanism as for Yocto

- The Ubuntu is generated from Yocto build, with Yocto/bitbake commands. This includes
  - Getting the Ubuntu packages
  - Building the NXP specific source code
Customer Flow

Start Here:
xpx.com->Flexera

Flexera:
- Documentation
- Yocto Binaries
- Quality package
- No Open Source Archive

GPU User-space Libraries

Non-Open Source support: Manually Download from Flexera
Example: GPU (Vivante)

Separate locations for Open Source code (CAF) and proprietary (Flexera)

Install Open Source Linux BSP using repo commands

Open Source components automatically downloaded from CAF

GIT repository for Open Source components
- Linux
- U-boot
- Yocto
- Drivers & Apps

$ repo init -u https://source.codeaurora.org/external/autobsps32/auto_yocto_bsp
$ repo sync

$ # Download and enable GPU support in Yocto
$ bitbake gpu
Service Oriented Architectures enabled by NXP’s Automotive Linux BSP
Highly Optimized Sensor Fusion
- Various sensor data streams: Radar, Vision, LiDAR, V2X
- S32V234 automotive vision and sensor fusion processor
- LS2084A embedded compute processor
- S32R27 radar microcontroller

High Performance per Power
- Up to 90,000 DMIPS at < 40 W
- Complete situational assessment
- Supporting classification
- Object detection and localization
- Mapping

Ease of Development
- ROS Space
- Open ROS Space Linux®-based system
- Programmable in linear C
- Easily customizable
- Development environment for mainstream vehicles

Security
- CSE and ARM® TrustZone® technology

Decision Making
- Global Path Planning
- Behavior Planning
- Motion Planning

NXP Automated Drive Kit
- Computing: NXP BlueBox 2.0
- Vision: Front Camera Software with MIPI CSI2 Camera
- LiDAR: Selection of Lidars supported
- RADAR
- Inertial Measurement Unit & Integrated GPS
- Operating System
- Middleware: ROS (Robot Operating System) Adaptive AUTOSAR
Virtual Ethernet

Standard Ethernet Linux interface

Implementation:
- Virtual Ethernet Network Device
- PCIe driver

BlueBox

$ ifconfig pcie0 192.168.1.1

$ ping 192.168.1.2
PING 192.168.1.2 56(84) bytes of data.
64 bytes from 192.168.1.2: time=0.34 ms
64 bytes from 192.168.1.2: time=0.35 ms

$ ifconfig pcie0 192.168.1.2

Interfaces in LS Linux
pcie0

Virtual Eth Instances
veth

PCIe Driver (RC)
S32V

Interfaces in S32V Linux
pcie0

Virtual Eth Instances
veth

PCIe Driver (EP)
RC

LS2

S32V

BlueBox

PCIe Communication
MPC-LS Vehicle Network Processing Chipset for Service-oriented Gateways

Heterogenous multi-core processing
- Real-time + high-performance applications

Automotive meets enterprise networking
- CAN FD, LIN, FlexRay™ interfaces
- Up to 10 Gigabit Ethernet with packet acceleration

End-to-end security from vehicle to cloud
- Embedded Hardware Security Module for cryptography and secure key management
Vehicle Service-oriented Gateway Enables Opportunities

The NXP MPC-LS chipset enables service-oriented gateways
SW Environment: ‘Potential’ Use Case

- Provides each container with own network stack (i.e. IP Addr)
- Prevent hypervisor bottlenecks.
  - HW support for Hypervisor, IOMMU, Virtual IRQs.
  - Careful assignment of I/O to Guests.

- Provides each SoA App with network abstraction using SOME/IP (service IDs)

- e.g. Connected services
- e.g. Safety services
- e.g. OTA Manager

- e.g. Netcom
- e.g. Connected services
- e.g. Safety services
- e.g. OTA Manager

- HLOS User Space App
- LXC Framework
- Adaptive AUTOSAR Framework
- ProvenCore OS
- ARM Trusted f/w

- Linux OS
- QNX OS
- RTOS App
- CAN Gateway

- XEN Hypervisor (Type 1)
- XEN Hypervisor (Type 1)
- Classic AUTOSAR OS

- CPU (e.g. Cortex-A)
- CPU (e.g. Cortex-A)
- CPU (e.g. Cortex-M/R)

- Provides each container with own network stack (i.e. IP Addr)
Automotive Linux BSP – Ready Solution for SOA Prototyping

Service-orientated Architecture (SoA) is driving change in SW architecture across the vehicle
Moving to scalable, abstracted platforms

Optimized BSP items and the communication path to better leverage NXP’s HW resources
(e.g. DDS Security offload using the Security and Ethernet)

Productize strategic SW items
Strategic partnership for Production ready solution