CAN configuration within Autosar

Dr.-Ing. R. Machauer, Bosch Engineering GmbH

AUTOSAR (AUTomotive Open System ARchitecture) aims to standardize interfaces between software application functions and further between application functions and basic software modules in ECUs (Electronic Control Unit). Independence from underlying hardware and this modular software design allows exchangeability of software functionalities amongst ECUs. The integration of functions from different suppliers is established through a virtual function bus. The mapping to a certain network topology (or to ECUs) is carried out after that step. In this paper configuration of a CAN stack and related basic software modules in system architecture development according to AUTOSAR is shown.

1 Introduction

Nowadays in automotive environment E/E systems comprise of many ECUs, which are connected via different bus systems (CAN, LIN, FlexRay, MOST, proprietary serial lines, etc.). In luxury cars the number of ECUs may even reach 70. They are classified into several system domains like powertrain domain, chassis domain, body domain, safety, infotainment, etc. Inter domain communication is established through gateways. The functional behavior of an ECU is specified by the Original Equipment Manufacturer (OEM), and implemented by the hardware supplier. Each ECU implements certain application functions, which interact in some way with other ECUs. It is a common approach, that ECU hardware manufacturers also supply application software with their ECU. With this approach the system integration medium is more or less the bus system the ECU is connected to. With such a large number of ECUs and interacting functions error detection within a system is a challenge for the system integrator. The driving forces for AUTOSAR are [1]:

- Exchangeability of functional modules (multiple suppliers)
- Manage increasing E/E complexity associated with growth in functional scope, which yields to improved quality and reliability of E/E systems
- Higher flexibility for product modification, upgrade and update
- Enable detection of errors in early design phases

The AUTOSAR partnership was established in summer 2003 [1]. First release of AUTOSAR specification was published in mid 2005. A second release followed during the first quarter of 2006 [2]. In chapter 2 a short technical overview of the AUTOSAR concept and its layer model is given. In chapter 3 the Basic Software (BSW) modules of communication software(COM stack) and the underlying layers are discussed in detail.

2 AUTOSAR concept

The AUTOSAR software architecture provides new design choices to software engineers building distributed, communication software systems. A key aspect of this architecture is a design abstraction called the AUTOSAR Virtual Function Bus (VFB) (rf. Figure 1). VFB allows the design of software systems without reference to target hardware. Hence during the design process the mapping to a specific hardware is shifted to later phase. The VFB is the collection of all communication mechanisms (plus some interfaces to the basic software) on an abstract level.

An AUTOSAR system is build from of a set of software components (SW-Cs). Interaction among SW-Cs is accomplished only through statically defined ports. Each port is assigned an interface type, defining the “way” of communication. Two communication paradigms are provided for SW-C ports, called sender-receiver and client-server communication; Sender-receiver provides message passing,

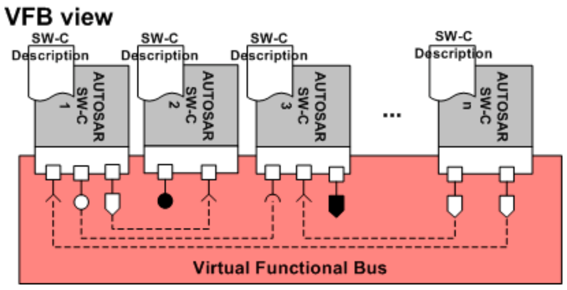

Figure 1: Virtual Function Bus [3]

whereas client-server communication invokes a service function call. VFB allows an early (virtual) integration of SWCs as soon as the communication mechanism amongst SW-Cs are defined. The AUTOSAR partnership specified a meta-model, described in the configuration language XML. Hence XML configuration files can be used for the entire system specification. These XMLconfiguration files are used as input for code generation tools, tools for creating object code or just as uniform specification exchange format. Additionally, development tools like bus analyzer, function modeling tools, etc. will support AUTOSAR exchange format in midterm.

The Run Time Environment (RTE) is the realization of the VFB for a specific ECU. The RTE realizes the communication between SWCs within an ECU (intra-communication) and among different ECUs (inter-communication). SW-Cs with their ports and interfaces are mapped to certain ECUs after the VFB system design step. XML configuration files are used to describe this mapping. After appropriate system configuration XML configuration files are fed to an RTE generator producing code (e.g. source code files rte.c, rte.h, etc.) specific to an ECU. In this way the RTE layer encapsulates the whole communication mechanism for SW-Cs by providing a standardizes API to application functions and simultaneously being independent from actual underlying hardware.

2.1 System layer model

A technical overview on AUTOSAR is given in [3]. This subsection focuses on the architecture within an ECU. Figure 2 depicts the software layer architecture. A

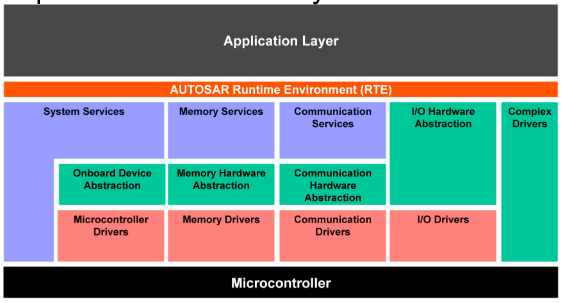

Figure 2: Software layer model [4]

color schema helps Figure 2: Software Layer Model [4] in distinguishing different layers. The application layer resides on top of the layer model. This is abstracted through the RTE from the underlying hardware (including ECU). The RTE layer is generated completely by tools based on the provided XML-schema. Developing such tools is not in the focus of AUTOSAR, but they play a key role. Below RTE all BSW-modules are divided into several layers. Each layer abstracts its adjacent underlying layers a certain degree more from real hardware. The service layer (blue) comprises system services (e.g. Operating System (OS)), memory services (e.g. Non Volatile RAM (NVRAM)) and communication services (like bus communication). An example of such abstraction would be the communication service layer, which abstracts from communication hardware. The network system (LIN, CAN, FlexRay) is still not known. The hardware abstraction layer, on the other side, supports interfaces, providing independence from hardware driver software and therefore from real microcontroller. On board device interfaces, memory hardware, I-O drivers and communication hardware modules are distinguished in the layer next to the microcontroller. The communication hardware abstraction layer hides the used network system. Through this module the network system type is chosen. For example a CAN interface module is used, if signals shall be transferred on CAN. Apart from these hardware abstraction blocks two more (green) BSW modules appear. Input/ Output (IO) hardware abstraction layer hides controller specific properties of input and output ports. Interfaces are provided to access data from ECU inputs or to put data to hardware output ports with physical values. The kinds of ports (Analog Digital Converter (ADC), Pulse Width Modulated (PWM), or digital input/output) shall not be of relevance to upper layers. The rightmost side of Figure 2 shows another BSW module Complex Device Driver (CDD). CDD allows direct access of RTE to peripheral hardware. This module is used for very time critical applications, which can be speed up through directly accessing peripheral hardware. Further supplier can protect their IP within a CDD.

3 COM stack

This section gives an elaborated overview of the COM stack. For simplicity only the COM stack for CAN communication is sketched. Figure 3 gives more insight into the layered architecture of COM related modules. The service layer contains several modules. Subsection 3.1 gives an overview of AUTOSAR COM, subsection 3.2 gives details on PDUrouter, subsection 3.3 gives an overview of CAN transport protocol (CAN TP), subsection 3.4 and subsection 3.5 summarize the task of CAN Interface and of CAN Driver modules.

3.1 COM

AUTOSAR COM is based on the OSEK COM specification [5]. The COM service layer provides a uniform interface to the CAN network. Protocol and message properties are hidden from the application. COM provides a microcontroller and ECU hardware independent interface to application. COM transfers signals to and from the RTE. Signals are packed into and from I-PDUs. Dependent on the service to be used, signals are directly passed to transport protocol modules or to the PDU Router. External signal exchange between SW-Cs on different ECUs are routed through RTE via COM to PDU-Router and then to a bus system as configured during system design. COM is able to work as a signal gateway, without inclusion of higher layers. Attributes are attached to signals, which are managed and evaluated within COM. Network management is included in the communication services layer apart

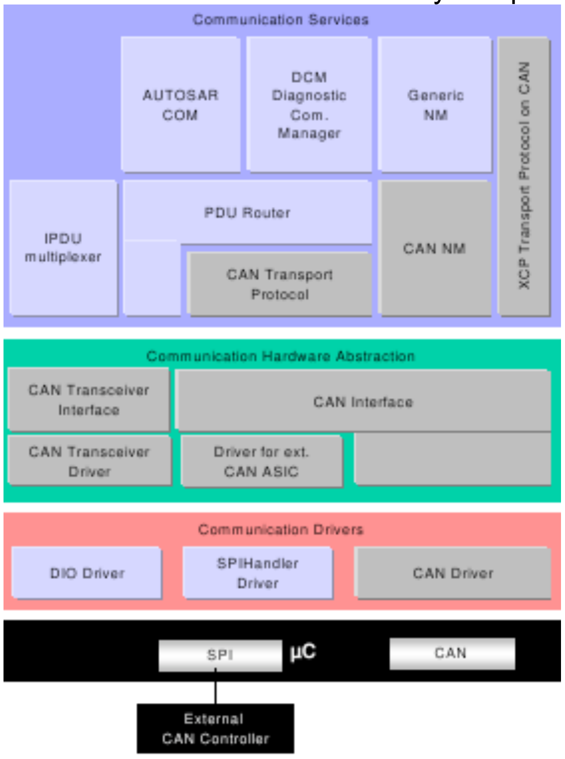

Figure 3: Communication stack layer model [4]

from AUTOSAR COM. Network management provides uniform services to get and set modes of networks nodes.

3.2 PDU-Router

The PDU-Router [6] provides services for routing I-PDUs between the following modules:

- communication interface modules (e.g. LIN, CAN and FlexRay)
- Transport protocol modules (e.g. CAN TP, FlexRay TP)
- AUTOSAR Diagnostic Communication Manager (DCM) and Transport Protocol (TP) modules (e.g. CAN TP, FlexRay TP)
- AUTOSAR COM and communication interface modules (e.g. LIN, CAN or FlexRay) or I-PDU Multiplexer
- I-PDU Multiplexer and communication interface modules (e.g. LIN, CAN or FlexRay)

The PDU router provides an API to the above layers (e.g. COM) and an API for modules below the PDU router (e.g. CAN interface). PDUs are identified by static PDU IDs. The routing operation of the PDU router is controlled by routing tables, which contain routing attributes for each PDU, e.g.: PDU Id and destination address. The PDU router does not modify I-PDUs, it simply forwards the I-PDU to the destination module. A detailed sketch of the PDU router structure is given in Figure 4. The PDU-R module is split into two parts:

- The PDU Routing tables
- The PDU Router engine

The PDU router engine performs routing actions according to the routing tables. Two translations can be distinguished: up (to higher layer) and down translation. In order to provide access to the DCM for the activation of the bootloader, the PDU router engine provides a minimum routing capability. Specific PDUs can hence be routed correctly to DCM in case the configurable PDU router tables are corrupted (e.g. by a previous flash operation).

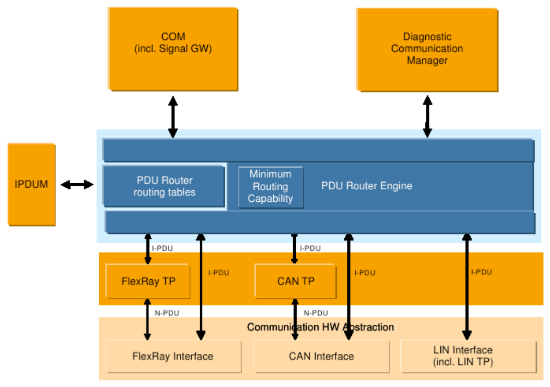

Figure 4: PDU router structure [6]

3.3 CAN-TP

CAN Transport Protocol (CAN-TP) is located between PDU router and CAN Interface (see Figure 3). CAN TP provides services for [7]:

- Segmentation of data in transmit direction
- Reassembling of data in receive direction
- Control of data flow
- Detection of errors in segmentation sessions

The main purpose of CAN TP is to segment and reassemble CAN I-PDUs longer than eight bytes (data flow control). Two routing paths exist between CAN interface and PDU router engine (see Figure 4). The PDU router determines whether a transport protocol is to be used or not. The PDU router can handle different communication protocols for COM and DCM I-PDUs. COM I-PDUs are generally routed directly to CAN interface, whereas DCM I-PDUs are routed via CAN TP, in a CAN network system. Diagnostic information [8, 9, 10] exceeds eight byte data fields, which makes data flow control necessary.

3.4 CAN interface

The CAN Hardware Interface provides uniform mechanisms to access a CAN bus channel regardless of its location (internal/ external), i.e. upper layers do not differentiate, whether a CAN controller is connected via an SPI bus or whether an on-chip CAN controller is used. CAN-Interface abstracts from the location of CAN controllers (on-chip/on-board), the ECU hardware layout and the number of CAN drivers. The upper layer addresses CAN channels rather than CAN controllers.

3.5 CAN driver

The CAN Driver provides services for initiating transmissions and callback functions for notifying receive events, independent from the CAN controller hardware.

3.6 CAN transceiver driver

The CAN Transceiver driver controls the external CAN transceiver hardware. It controls wake-up and sleep of the CAN bus. It observes the bus line and provides physical network layer diagnostic information (short circuit, open line, etc.) to the upper layers.

4 Example application

In this section the configuration and integration process of a small mirror demo application is illustrated. First the demo application is split into SW-Cs: SensorInputProcessing, MirrorAdjustManager, ActuatorOutputProcessing. These SW-Cs communicate with each other over ports and their assigned interfaces. Figure 5 depicts the top level design. A small mirror control application serves as a demonstration. The example system consists of three SW-Cs. The left SW-C (SensorInputProcessing) reads in sensor information provided by the car driver via joystick movements. The right SW-C (ActuatorOutput- Processing) sends steering commands to actuators. The MirroAdjustManager SW-C coordinates incoming requests and calculates corresponding move commands. At this step sensor- and actuator component depends on logic signals, like: MoveX, MoveY, Stop, etc. Now ports, interfaces and data elements are defined. Table 1 shows an extract of a software component XML description. For simplicity a table is used instead of copying XMLcode. Table 1 is related to the leftmost SW-C of Figure 5. The name of the atomic SW-C is defined in the first row. The component contains a providing port (p-port) PP_Req_Mirror, which uses a sender-receiver (S/R) interface of type IF_MirrorMoveXY. The communication type sender-receiver involves transmission and reception of signals for atomic data elements. An S/R interface may contain multiple data elements. Here DT_Req_MoveX and DT_Req_MoveY are two data elements

Parameter	Settings
Atomic-Software-Component-Type	CT_SensorInput-ProcessingType
P-Port-Prototype	PP_Req_Mirror
Sender-Receiver-Interface-Type	IF_MirrorMoveXY
Data-Element-Prototype	DT_Req_MoveX DT_Req_MoveY

Table 1: SensorInputProcessing configuration parameters (extract)

defined within the interface IF_MirrorMoveXY. These data elements are written by the pport. A require port (r-port) represents a read data element

operation, as can be seen at the rightmost SW-C in Figure 5. Graphical tools will be (soon) available to configure SW-Cs by drawing them as they are sketched in Figure 5. These tools support import and export of XML code, in order to exchange configurations with other tools. In the next development phase SW-Cs are mapped to hardware (ECUs), which represents mapping to a certain system architecture design. Tools also assist in this step. A manual way is, to map the SWCs by means of AUTOSAR XML code. After this step intra ECU and inter ECU signals are distinguished for the first time. For example:

If SensorInputProcessing is mapped to ECU1 and the right SW-Cs MirrorAdjust-Manager, ActuatorOutputProcessing are located on ECU2, the ports PP_Req_Mirror and RP_Req_Mirror exchange their data elements between ECUs. Data elements between the ports PP_cmd_Mirror and RP_cmd_Mirror are intra ECU signals. In case no special routing is necessary, nothing is to be done for intra-ECU signals. All intra ECU traffic of SWCs is managed by RTE. Inter ECU signals are communicated to external devices. They are passed from RTE to COM and vice versa. COM packs signals into I-PDUs, which contain one ore more signals. Signals within a COM I-PDU must occupy contiguous bits. Signals in COM layer need a reference to data elements of SW-C ports. Data elements of SW-C ports has to be mapped to I-PDU signals. Through this mechanism data elements from SW-C ports are linked to external networks (a CAN network in this case). While transmitting the PDU-Router reads the I-PDUs and routes them according to the routing table to lower layers. In case of reception PDU-R transfers I-PDUs without modifications to upper layers.

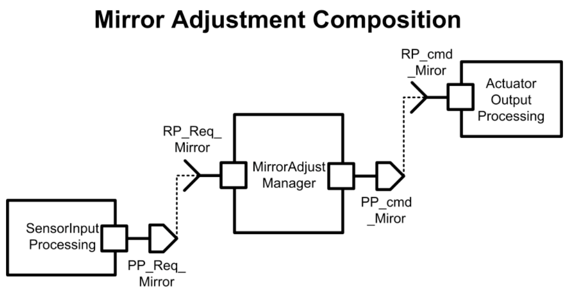

Figure 5: SW-Cs of demo application

5 Conclusion

The AUTOSAR standard aims to achieve standardized interfaces between software application functions and from them to basic software modules within ECUs (Electronic Control Unit). Software modules designed this way, are independent from their underlying hardware. Software modules may be exchangeable due to a standardized API to other modules or to application functions. In future automotive system architectures the number of ECUs may be reduced, because of the ability to integrate SW-Cs on a virtual function bus in a quite early development phase. The increasing functional scope with the growing complexity may be kept feasible in future systems. The architectural concept of AUTOSAR is based on a VFB. Per ECU the VFB is instantiated as an RTE layer. RTE encapsulates the whole communication infrastructure for SW-Cs which build the application functions. BSW modules are standardized for configuration. Also BSW module APIs are part of the AUTOSAR standardization, which yields exchangeable BSW modules (e.g. from different suppliers). Each layer abstracts gradually its underlying layer from hardware. Data elements of SW-Cs are identified as internal or external signals when they are mapped to certain ECUs according to a system network architecture. External signals are transferred through RTE to COM, where they are packed/ unpacked into/ from I-PDUs. User data is put into (or get from) buffers of the PDU-router, which routes the packed I-PDUs according to its routing table. A default routing table exists for diagnostic data exchange, which serves as backup in case a prior program sequence destroyed the routing table. The router selects to (or from) which network interface data is transferred. In case of CAN the I-PDUs are put (or get) to the CAN Interface, which transfers them finally to the CAN driver and from there to the CAN controller.

6 Acronyms, abbreviations

ADC	Analog Digital Converter
BSW	Basic Software
CAN-TP	CAN Transport Protocol
CDD	Complex Device Driver
DCM	Diagnostic Communication Manager
NVRAM	Non Volatile RAM
OEM	Original Equipment Manufacturer
PWM	Pulse Width Modulated
RTE	Run Time Environment
OS	Operating System
TP	Transport Protocol
VFB	Virtual Function Bus

References

[1] H. Heinecke, K.-P. Schnelle, H. Fennel,J. Bortolazzi, L. Lundh, J. Leflour, J.-L.Mate, K. Nishikawa, and T. Scharnhorst, AUTomotive Open System ARchitecture -an industry-wide initiative to manage the complexity of emerging automotive e/earchitectures. Convergence 2004, October 18-20 2004.

[2] AUTOSAR. AUTOSAR partnership homepage: www.autosar.org.

[3] AUTOSAR. Technical Overview, 2.0.0 edition, March 2006.

[4] AUTOSAR. Layered Software Architecture, 2.0.0 edition, March 2006.

[5] OSEK. OSEK/ VDX Communication, 3.0.1 edition, January 2003.

[6] AUTOSAR. Specification of PDUR, 2.0.0 edition, April 2006.

[7] AUTOSAR. Specification of CAN Transport Layer, 2.0.0 edition, April 2006.

[8] ISO 15765-2 (2004-10-12), Road vehicles - Diagnostic on Controller Area Networks (CAN) - Part2: Network layer services.

[9] ISO 15765-3 (2004-10-06), Road vehicles - Diagnostic on Controller Area Networks (CAN) - Part3: Implementation of Diagnostic Services.

[10] ISO 15765-4 (2005-01-04), Road vehicles - Diagnostic on Controller Area Networks (CAN) - Part4: Requirements for emission-related systems.