Steer-By-Wire System Development Using AUTOSAR Methodology

Khaled Chaaban, Patrick Leserf and Sébastien Saudrais
ESTACA, Embedded Systems Laboratory
Rue georges Charpak BP76121 - 53061 LAVAL Cedex 9
firstname.lastname@estaca.fr

Abstract

This paper presents the development of a Steer-By-Wire system using AUTOSAR methodology. AUTOSAR defines common standards for the development of embedded automotive software. Some aspects of safety and timing requirements are analyzed. We show how AUTOSAR model permits to design and implement a safety embedded system using well define steps. First, the system is validated by simulation and then a partial implementation of the system is performed on an embedded target for real tests.

1. Introduction

In order to improve safety and to provide new and emergent functionality in vehicles, car manufactures have increased the use of electronic control systems and have encouraged the introduction of Drive-By-Wire systems [3]. An example of such systems is the Steer-By-Wire system. The system must fulfill both safety and real-time requirements. Such system is considered as time-critical and the timing properties impact directly the car safety. For example, the delay between a driver’s request and the activation of the actuators wheels can be considered as a performance indicator of the system. If it exceeds a certain limit; the driver could totally loss the control of his car.

The complete design of such distributed embedded system that includes several functions can be complex involving a multitude of steps dealing with communications, control, and computing issues.

AUTOSAR consortium has been founded to facilitate the integration of third party software and supplier subsystems, the easy reuse of software and hardware components and seamless application of diverse development tools. AUTOSAR concept is based on a model-driven development process by using a standardised formal specification model for the automotive software structure. The software architecture, as well as the ECU (Electronic Control Unit) hardware and the network topology, are modeled in a formal way, which is defined in a metamodel that supports the software development process from architecture up to integration. All available modeling elements are specified by the “AUTOSAR metamodel” [5].

However, AUTOSAR specifications does not address the timing constraints adequately now. This is an important factor to be considered for a successful application of the standard and for its continuous deployment within the automotive industry. It is envisaged to integrate the timing behaviour and constraints in the future AUTOSAR specifications.

This paper deals with the development of a Steer-By-Wire system using AUTOSAR methodology. There are few works in this context. In [7], authors propose safety embedded architecture for the development of a Steer-By-Wire system using CAN network. Their hardware architecture is very similar to our architecture. Their developed system was simulated using CANoe tool and then validated on real target using Microchip solutions for ECUs and CAN interfaces. Their works has focused on safety design analysis. In [4], authors propose an integrated method for evaluating both real-time performance and “Behavioural Reliability” of a Steer-By-Wire system. They define a QoS indicator of the system that tries to optimize it. Their results are validated on a real embedded architecture with both TT and Flexray networks. To our knowledge, there are no works concerning the development of a Steer-By-Wire system using AUTOSAR model. Little industrial information is published in the literature concerning the development of a system prototype but without details.

The rest of the paper is organized as follows: section 2 presents the embedded steering system while describing functional view, hardware view and safety requirements. Section 3 gives a brief overview on AUTOSAR concept before the description step-by-step of the system development process using AUTOSAR methodology. The section identifies some timing requirements of the steering system that must be supported by AUTOSAR model. Section 4 presents some implementation keys with real tests and validation of the system.

2. System description

2.1. System overview

A basic steer-by-wire architecture composed by three main blocks: the hand wheel (i.e. steering), controllers and the road wheels as illustrated in Figure 1.
At the steer side, two kinds of sensors are necessary to acquire the steer angle and the torque applied by the driver. In addition, a feedback actuator is necessary also at the steer side so that the driver get the feeling of turning a traditional steering wheel and feels the effect of tuning the wheels on a certain type of road.

At the wheel side, the sensors are considered of the same type and they provide the same information type. In addition, an actuator is used to control the steering rack.

The distributed controllers are connected using a Flexray network. Sensors and actuators are connected directly to the I/O ports of controllers.

2.2. Functional view

The Steer-By-Wire system is composed of two main functions: (1) the feedback torque function and (2) the rack torque function (Figure 2).

The feedback torque function is essential for the system operation. It computes the feedback force applied to the steering wheel so that the driver feels the effect of tuning the wheels on a certain type of road.

The feedback torque is computed using a realistic order law using pacejka extended model [6] and depending on wheel torque, steer torque and the vehicle speed (driving situation factor). This function is modeled using Matlab/Simulink and the corresponding code is generated using RTW of Matlab.

Figure 2. Two main functions of the steering system.

The rack torque function is the main system function that permits to control the front axle actuator. It is computed using a discrete PID controller with sampling time of 1 ms.

2.3. Hardware view

We describe by this section a safety hardware architecture augmented by certain mechanisms of fault-tolerance and recoverability in order to achieve the fail-safe requirements by replicating system components.

Figure 3 illustrates the hardware architecture of the entire system. The architecture is composed of 4 ECUs and it seems to be a realistic one for commercial vehicles. The architecture is composed of three steer sensors (SS) connected to two steer manager ECUs, a Flexray network with two duplicated channels (channel A and channel B), two wheel manager ECUs connected to three wheel sensors (WS). Steer manager ECUs are also connected to two steer actuators for the feedback torque, and wheel manager ECUs are connected to two wheel actuators to turn the wheels.

Figure 3. Hardware architecture.

2.4. Safety requirements

The main objective for introducing a steer-by-wire system is to improve the vehicle safety. The system is therefore classified as safety-critical and it must continue to operate in the event of a fault and the risk must be minimized. In this section, we present some aspects of safety requirements, fault-tolerance and partial analysis of the system dependability is presented. The proposed methods are validated by simulation but without being proved formally as it is not the paper’s topic.

2.4.1. Operation modes of a manager ECU

For each manager ECU, we have defined three operation modes: "OK", "WARNING" or "EMERGENCY" mode as shown in Figure 4.
Switching between replicated ECUs is done at runtime regarding the states of connected sensors and actuators. A priority value is attributed to a manager ECU to specify it as the primary replica in case of identical states values.

Moreover, the manager ECU state is strongly linked to the states of its connected sensors and actuator.

2.4.2. Sensors and actuators management

Figure 7 (see next page) illustrates our proposed algorithm that permits to determine the state of sensors. The algorithm starts by testing the number of sensors whose values are within the predefined data ranges. Then it computes predefined tolerance intervals using pair-wise comparisons between preselected sensors. Finally, it classifies sensors’ state according to the number of valid sensors. For actuator, we have only two states: “OK” and “EMERGENCY” since we have one actuator per manager ECU.

<table>
<thead>
<tr>
<th>State</th>
<th>Sensors</th>
<th>Actuator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ok</td>
<td>0</td>
<td>Ok</td>
</tr>
<tr>
<td>Warning</td>
<td>1</td>
<td>Emergency 2</td>
</tr>
</tbody>
</table>

Figure 5. Sensors and actuators states.

2.4.3. ECUs states management

The redundancy of ECUs is done in a semi-active manner. i.e. replicated ECUs (steer and wheel manager) receive and then treat the synchronously but only one ECU is “activated” (primary replica) at a given instant and the second is “inactivated” (secondary replica) as shown in Figure 6.

"Inactivated" state means that the manager ECU is in standby mode or failed. It continues its execution in background and sends its computed values on the FlexRay network but it does not command the corresponding connected actuator.

"Activated" state means that the manager ECU is active and it also commands the connected actuator.

Figure 6. States of a manager ECU.

Now that we have determined the sensors and actuators states, we can determine the ECU state.

In the same way, we can determine the state of the Wheel Manager ECUs.

3. Development process using AUTOSAR methodology

This section begins by a brief presentation of AUTOSAR concept and then we present the development process of the embedded steering system using the well defined AUTOSAR steps.

AUTOSAR (AUTomotive Open System ARchitecture) is introduced by automobile manufacturers, suppliers and tool developers as the future standard of automotive E/E engineering. By breaking up the cohesion between hardware infrastructure and the application software, embedded automotive system complexity can be managed and software reuse is promoted.

AUTOSAR development methodology is based on a model-driven development style. The software architecture, as well as the ECU hardware and the network topology, are modelled in a formal way, which is defined in a metamodel that supports the soft-ware development process from architecture up to integration. All available modelling elements are specified by the “AUTOSAR metamodel” [5][2]. The metamodel is defined according to the OMG Meta Object Facility [1].

An AUTOSAR system consists of software components (SWCs) communicating and interacting through a Virtual Functional Bus (VFB). SWCs are then mapped to specific control units distributed (ECU) over a network. As the result of a layered architecture, they can be transferred to other platforms without detracting from the individual functions.
Yet, AUTOSAR is still facing many problems and obstacles. The dependency inside software, the dependency to hardware, development of subsystems by different teams, and the strict timing constraints and timing properties are still not totally covered by actual AUTOSAR specifications.

Figure 7. Sensors arbitration algorithm.

Today, timing is mostly taken into consideration late in the AUTOSAR development process, during the implementation and integration phase. Timing behaviour is verified by means of measurements at testing time, rather than through formal and systematic analysis. The likely consequences are long and costly design iterations whenever problems are detected. For this reason, a European research project called TIMMO (TIMing MOdel) was created to define a common, standardized approach for handling all timing-related information during the AUTOSAR development process [8]. These works are planned to be integrated in future release of AUTOSAR metamodel (4.0).

Next, we detail these steps through our case study of an embedded steering system. Let’s note that the figures are sometimes simplified to avoid highly connected illustrations.

3.1. A. SWC description and VFB view (step 1, 2)

The first step of AUTOSAR development process defines the set of SWCs constituting the software architecture of the embedded steering system and using the functional decomposition described above.

Software components communicate using ports and their interfaces. An interface may be of sender/receiver, client/server or calibration type. Further, a SWC may be one of the three types: Sensor/Actuator, Application or Calibration type. In our case study, we use only Sensor/Actuator and Application SWCs.

Figure 8 shows the software components structure of the embedded steering system. Only functional connections are illustrated to avoid a highly connected figure. Interconnections between components are also omitted (Inter ECUs connections) for simplification.

Figure 8. AUTOSAR development process.

In general, the development process of an AUTOSAR application passes through 6 steps as shown in Figure 8.
Communication interfaces of SWCs are of sender/receiver type. As shown in Figure 9, we have one SWC per steer or wheel sensor (tripled), one SWC per steer or wheel actuator (duplicated) and then one application SWC for steer or wheel manager (duplicated).

Figure 9: software architecture

Figure 10. VFB view of the rack torque function.

Figure 10 shows the simplified signal flow and involved components of the rack torque function. It has been identified that the signal path of execution will affect four components. The SteerSensor component acquires the sensor physical data and passes it to the application software component SteerManager for treatment. Afterwards the signal is sent to the application software component WheelManager for order computation until it is finally send to the actuator via the WheelActuator component.

At this level, we neglect the physical distribution of components and the underlying network and thus allowing a logical representation of the entire system with validation of functional features of main functions.

3.2. Internal behaviour of software component (step 3, 4)

Now that the SWC type and external interfaces are defined, we must describe the internal behaviour of each SWC that was represented as a black box in the VFB view. The internal behaviour decomposes a SWC into runnable entities, which are executed at runtime.
As an example, Figure 11 shows the inside runnables of the WheelManager component. The execution of a runnable is triggered by an event. An event may be of timing or data type. For the WheelManager component, we have eight runnables, three timing events (blue colour) runnables and five data event (burn colour) runnables. We consider only periodic semantics for events triggering. Timing event-based runnables are triggered at the period of 5 ms. Data event-based runnables are triggered at the arrival pattern of data from connected sensors or other components of the system (inter ECUs).

The three timing event runnables are:
- Run_Sensors determines sensors’ status using the arbitration algorithm described before for both angle and torque measure. It provides also a single sensor value (best one) of angle and torque to “Run_Command” that used it without further treatment.
- Run_State computes the state of WheelManager component according to the status of other application components, the state of its sensors and the state of its actuator. The objective is to provide information about the component’s status to runnables who need it for their treatment.
- Run_Command computes the rack torque to be applied to the front axle actuator. It returns also to the steer manager components information on the angle and torque at the wheel.

Runnables exchange data via specific AUTOSAR variables called inter-runnables variables instead of global variable that is prohibited in AUTOSAR specifications.

3.3. System description (step 3, 4, 5)
An AUTOSAR software component must be independent from any ECU. In contrast sensor and actuator software components are bound to an ECU, exactly to that ECU where the sensor or the actuator is connected to (Figure 12).

3.4. Basic Software (step 6)
In AUTOSAR architecture, the Basic software (BSW) makes the link between RTE and all hardware features of the microcontroller (µC). BSW is composed of 80 modules abstracted by 3 layers: the service layer, the ECU abstraction layer and the µC abstraction layer.

Figure 11. Runnables of the WheelManager component.

Figure 12. SWCs mapping.

The service layer provides µC and ECU independent services like Operating System (OS) and
communication services. In our application and during OS configuration of WheelManager component, we have defined one task for the state and command runnables, one task for sensors runnable and one task for interECU runnable linked to FlexRay with higher priority.

For the communication services, we configure the inter-ECU communication, independently of the possible low level bus (CAN/FlexRay or LIN). In our case, the wheel manager has to collect angle and torque signals (4 bytes length) from each rack sensor, group them into signal group, packing them into PDU (protocol data unit). Then a PDU router distributes PDU into specific protocol frame (FlexRay frame here), with packing/unpacking service.

3.5. Timing requirements

A Steer-By-Wire system must fulfill both reliability and real-time constraints. Such system is considered as time-critical and the timing properties impact the car safety. For example, the delay between a driver’s request and the activation of the wheels can be considered as a system performance indicator. If it exceeds a certain limit, the driver could totally loss the control of his car. So, the reliability is not sufficient and the real-time performance should be evaluated.

We present by this subsection some relevant timing properties and constraints that we could identify during the system development process and that must be supported by AUTOSAR metamodel.

As cited before, the main timing constraint of a Steer-By-Wire system is the response time of the rack torque function, i.e. the pure delay introduced between the steer request to the reception of the driver’s request by the rack actuator. This response time can also impact the availability and in the worst case, the safety of the system.

Below, we show how this timing constraint can be mapped at different levels of the system development.

3.5.1. VFB-level timing

At the VFB level, the main timing constraint that we can identify is the end-to-end timing constraint with physical sensors and actuators (Figure 13).

This timing constraint is considered as reactive delay since its age impacts the system QoS. This is a high-level application requirement and must be announced early in the design phase using AUTOSAR metamodel.

3.5.2. SWC-level timing

At this level we illustrate as an example the two scenarios of synchronization and execution order of the WheelManager SWC.

As shown in Figure 11, the WheelManager SWC is composed of a set of runnables that are executed concurrently with some precedence constraints. For example, ‘Run_State’ uses the results of ‘Run_Sensors’ to compute its outputs. Likewise, ‘Run_Command’ uses both runnables to act on the rack actuator. So, the execution of runnables must be synchronized and scheduled at the correct rates to have a correct behaviour. Moreover, the synchronization between sensors must be ensured. For our application, we use triple sensors for wheel and steer information and we must define a maximum difference between sensors sampling events at the input of sensors runnable. The synchronization of actuators is not mandatory in our case since we use a semi-active replication of ECUs.

3.5.3. System-level timing

In this level, we can use the same timing elements of VFB timing and SWC timing levels but with different context. The difference is due to the software mapping of SWCs to ECUs and the definition of communication cluster.

The driver request information is transmitted from the steer sensor to the wheel actuator through different nodes via FlexRay network. Sensors and actuators are directly linked to an ECU via point to point links.

This timing constraint implies the use of end-to-end timing chains and sub-chains of the AUTOSAR timing extensions in order to make a holistic scheduling analysis of the whole distributed system. The sensors and actuators delays must be provided by the high level model in order to make a complete timing analysis.

As shown in Figure 14, the end-to-end timing chain is composed of five timing chain segments. Using BSW and communication stack, we can determine the maximum execution time of each timing segment.

4. Test, validation

In order to implement and then validate our embedded steering system, three phases of tests are performed.

The first phase concerns the unit testing of runnables, the smallest executable entities of the AUTOSAR system. In particular, we test here the developed algorithms of each runnable (Run_Status, Run_Sensors and Run_Command).

Now that the runnables are tested individually in simulation with RT builder tool, the second phase is the integration tests. We test the distributed functions (feedback torque and rack torque functions) with consideration of all the relevant software components and inter-connections.
5. Conclusion

In this paper we have presented a Steer-By-Wire system development using AUTOSAR methodology. We have shown how AUTOSAR facilitates the composition and then integration of complex embedded system using well defined steps. Some key elements of safety and timing requirements are identified during system design. The system is then validated by simulation and then by real implementation for real tests. The experimental platform is being built at ESTACA, with different AUTOSAR tools.

References