Composability and Compositionality in CAN-Based Automotive Systems based on Bus and Star Topologies

Roland Kammerer, Bernhard Frömel
Vienna University of Technology, Austria
{kammerer, froemel}@vmars.tuwien.ac.at

Roman Obermaisser
University of Siegen
roman.obermaisser@uni-siegen.de

Paul Milbredt
AUDI AG, Germany
paul.milbredt@audi.de

***Abstract*—Controller Area Network (CAN) is the most widely used field bus protocol in the automotive domain. The development process of today's cars follows the well established automotive V-Model. Traditional bus-based CAN makes the development an ever increasing challenge. For example, the introduction of a single additional CAN message influences the timing of already existing messages and thereby increases testing and integration efforts. The lack of composability and compositionality of traditional CAN leads to an overhead in the whole development cycle. In this paper we propose a development process that is based on a time-triggered CAN router. We examine the influence of our proposed development approach on major phases of the automotive V-Model. Our evaluation is based on CAN traffic of a mass-produced car by a major car manufacturer and a Fiel Programmable Gate Array (FPGA) based prototype implementation of the CAN router. From the results we gathered during our evaluation we conclude that a CAN router based development approach has the potential to simplify the development efforts that have to be undertaken by car manufactures.**

I. Introduction

Controller Area Network (CAN) [11] is the most widely used communication protocol in the automotive area. Present day cars contain several CAN networks with bandwidths ranging from 125 kbit/s to 500 kbit/s as part of the powertrain, safety, body and comfort domain [16].

The widespread use of CAN resulted from its protocol features aimed at performance, robustness and flexibility at low cost [25]. CAN offers multicasting by message filtering, remote data requests and consistent message delivery based on restrictive fault assumptions [12]. The non-destructive contention-based arbitration in CAN enables designers to control the timing of the CAN bus using message priorities. Temporal guarantees can be established using restrictions for the transmission behavior and timing of the application software (e.g., periodic messages, minimum interarrival times) [24]. CAN also includes protocol features targeting robustness including error detection features, automatic retransmission of messages, distinction between temporary errors and permanent failures of nodes and autonomous deactivation of defective nodes.

With the increasing criticality and more stringent real-time requirements of in-vehicle applications, substantial deficiencies of CAN have been highlighted with respect to composability, scalability, reliability and real-time performance [14]. These deficiencies lead to automotive protocols such as FlexRay [4] and Time-Triggered Ethernet [21]. These protocols avoid the limitations of CAN at increased cost of the communication infrastructure (e.g., more expensive cabling, additional hardware for media replication and communication controllers). In addition, CAN-based legacy applications require costly redevelopment and retesting for the migration to a new communication infrastructure. With present-day cars including up to a hundred million lines of code [5], the complete and instantaneous migration of existing automotive application software has become infeasible.

In prior work, new CAN-based communication infrastructures based on active star couplers [3], [13] were introduced that aim at improving the fault-tolerance, composability, scalability, reliability and real-time support for legacy applications. Due to backward compatibility and full compliance to the CAN standard, these communication infrastructures avoid costly redevelopment and revalidation.

This paper experimentally evaluates the behavior of a legacy application when using the ACROSS platform [19] based time-triggered CAN router, which belongs to the family of star couplers. We perform an evaluation based on the real-world traffic from the powertrain domain of a series car. We compare a bus-based CAN system with a system using the CAN router. In particular, composability and the stability of components services upon integration are evaluated by exploring the changes in the temporal behavior upon the successive addition of nodes.

The results show that the worst-case and average behavior is improved, which is beneficial for the majority of CAN-based applications including cyclic control functions. However, the best-case behavior deteriorates due to the overhead introduced by the CAN router. Due to the limited number of router ports our setup allows nodes sharing the same CAN bus, which is then connected to a router port. Therefore, the router does not establish complete temporal isolation as performed by time-triggered communication protocols [4]. The presented experimental results highlight the fundamental limits with respect to fault-tolerance without changes to legacy applications.

The paper is organized as follows: In Section II we state the requirements necessary for composability and compositionality. In Section III we give a brief overview about the current development process in the automotive domain. Continuing from that, we describe the integration process of Electronic Control Units (ECUs) in Section IV. The focus is on a comparison between existing bus-based CAN and star couplers, especially the time-triggered CAN router. In this

section we also discuss the influence of using the CAN router on the existing development model used in the automotive domain. In Section V we present our evaluation approach consisting of our test environment, the test cases and the results we gathered. These results get discussed in Section VI, and finally, Section VII concludes the paper.

II. Requirements for Composability

Composability is a concept that relates to the ease of building systems out of subsystems [1, p. 24]. An architecture supports composability if it supports the constructive composition of large systems out of components and subsystems without uncontrolled emerging behavior or side effects. An example is the deadlock-freedom of a component-based system.

Compositionality is a related concept that refers to the ease of validation of system properties. In a compositional architecture, properties of the system can be inferred solely by the analysis of related properties of its components. This means that the analysis of the component properties can be performed independently for each component. Compositionality avoids exponential analysis complexity. Compositionality is necessary for correctness-by-construction of component-based systems. Temporal compositionality is an instantiation of the general notion of compositionality. A communication system supports temporal compositionality, if *temporal correctness* is not refuted by the system integration [15].

A necessary condition for temporal compositionality is that if n nodes are already integrated, the integration of node $n+1$ will not disturb the correct operation of the n already integrated nodes. This condition guarantees that the integration activity is linear and not circular. It has stringent implications for the management of the network resources. Time-triggered communication systems satisfy this condition, because the allocation of network resources occurs at design time, prior to the system integration.

In an event-triggered communication system, however, that manages the network resources dynamically, it must be ascertained that even at the critical instant, i.e., when all nodes request the network resources at the same instant, the specified timeliness of all communication requests can be satisfied. Otherwise failures will occur sporadically with a failure rate that is increasing with the number of integrated nodes. When communication resources are multiplexed between node computers, then each newly integrated node computer will affect the temporal properties of the communication system (e.g., transmission latencies, bandwidth, jitter) of the message exchanges of already integrated nodes.

If a new node – from now on denoted as ECU – of a standard CAN system [11] is integrated and sends a message with a higher priority than a message sent by a previously existing ECU, then the message transmissions from the previously existing ECUs can be delayed. Delays occur, when the previous ECUs loose in the arbitration process when competing the newly integrated ECU. Even in case the new ECU sends only lower priority messages, the non-preemptive CSMA/CA protocol of CAN will cause a worst-case delay of a complete transmission duration of a lower priority message.

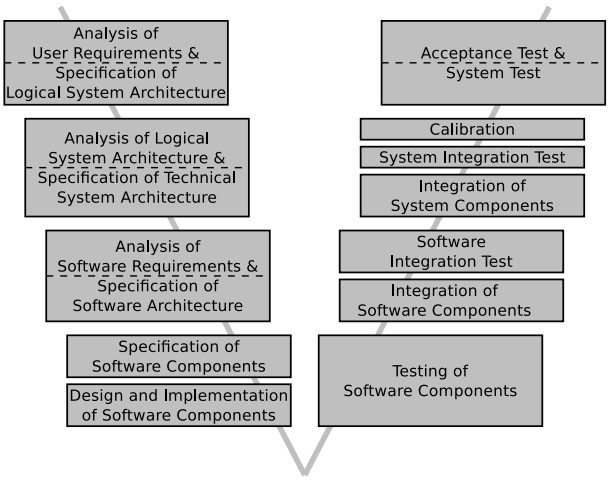


Fig. 1: Automotive V-Model

III. Development Process in the Automotive Domain

Over the last decades cars evolved from relatively simple systems like the Volkswagen Beetle to one of the most complex consumer goods produced. Main factors that led to that situation are the ever increasing demands on functionality like safety (e.g., Anti-lock Braking System (ABS), Traction Control System (TCS), airbag), comfort and fuel efficiency. Additional factors include the deployment under a harsh environment, high reliability, availability and a long life cycle of about 20 to 25 years [20]. Development of modern cars usually takes 3 years of the overall life cycle. In order to handle the complexity of vehicle development, the overall functionality of a car is split into several subsystems:

- Powertrain: Motor/engine ECUs, transmission ECUs,...
- Chassis: Wheels, breaks, steering, shock absorbers,...
- Body: Power window, wipers, heating, cooling. lighting, parking,...
- Multimedia: Radio, CD/DVD player, navigation system, telephone, Internet access,...

In previous times the subsystems were more self-contained and interacted on a lesser degree with each other. In modern vehicles this assumption does not hold as functions like the TCS require information from several subsystems (i.e., powertrain and chassis).

In the automotive domain the supply chain is traditionally split into so called tiers, which are structured in levels. Tier 1 suppliers for example have the competence to integrate complex subsystems, where tier 2 suppliers usually provide simpler subsystems or modules. The subsystems are then assembled by Original Equipment Manufacturers (OEMs) that have vehicle integration competence. Usually OEMs specify automotive functions and suppliers implement them.

Component development is usually driven by the so called *Automotive V-Model* [20], which is similar to the V-Model [8] known from other disciplines like software engineering, but has been extended to fit the needs of the automotive domain. Figure 1 shows this well established model. Basically, the model is subdivided into three phases: Design, Implementation and Test. In the following we sketch the individual steps of the automotive V-Model as described in [20]:

Analysis of User Requirements and Specification of Logical System Architecture: In this phase the often informal user requirements are transformed to a more formal and therefore more unambiguous requirements specification. From this specification it is then possible to derive a *logical* system specification. The logical system specifications provides a high level overview about required functions, required subsystems and their interfaces, while consciously leaving out details about the concrete implementation (i.e., the technical system architecture).

Analysis of Logical System Architecture and Specification of Technical System Architecture: In this step the logical system architecture that was derived in the previous phase is now mapped to a concrete technical system architecture. Steps include the analysis of real-time systems, of distributed networked systems and reliability and safety considerations. After the analysis is complete, a technical system architecture can be derived.

Analysis of Software Requirements and Specification of Software Architecture: The previous step specified the technical system architecture which can be used as the underlying platform for the current step. Software requirements are analyzed and a software architecture gets derived. This specification includes software components and their interfaces (e.g., data and control interfaces) as well as required software layers. In the automotive domain OSEK [10] and AUTOSAR [9] are well known and provide standardized software architectures. OSEK is an open standard developed by a consortium of German and French automotive manufactures and the University of Karlsruhe. OSEK's goal is the specification of three areas: Communication, operating system and network management [18]. AUTOSAR is a partnership of car manufactures, suppliers and companies in the electronics, semiconductor and software industry [2]. It extends and reuses the ideas of OSEK (e.g., backwards compatible operating system) and provides a standardized automotive software architecture.

Specification, Design and Implementation of Software Components: As the software architecture is specified, the specification phase concentrates on further specifying every software component of the software architecture. All software components are described with a data model, a behavioral model and a real-time model. A standard tool used in this phase is ASCET [7]. Nassi-Shneiderman diagrams [26] can be used for control flow representation.

Software Component Testing: Software testing is a researched field on its own and will not be discussed in detail in the context of this paper. Usually, well known techniques like peer reviews, static analysis, white and black box testing and various code coverage metrics are used in the automotive domain. Test cases used in this phase have been derived in the specification and design phase.

Integration of Software Components and Software Integration Testing: In the first part of this phase the software components that were potentially developed by different partners have to be integrated. Besides from creating the final software component, this step also includes generating the necessary documentation and description of tools (e.g., compiler, flash tools, parameters, ...). Integration testing includes for example checking naming conventions for variables and tests executed by the compiler (e.g., standard conformance).

Integration of System Components, System Integration Tests and Calibration: In previous steps software components were developed, tested and deployed to ECUs. In this step components that make up the system are further integrated. It is a horizontal process where for example ECUs are integrated to subsystems and subsystems are then integrated to the overall system, the car. Integration of system components has to be accompanied by testing the newly formed subsystems and systems. Calibration of the overall system is a step that can often only be done at a very late state of the development (i.e., fine tuning of the developed car after all components are integrated).

System and Acceptance Test: The System and acceptance test is the final step of the automotive V-Model and is part of the validation process. Previous development steps should guarantee that the system was developed according to its specification. The final acceptance test can only be done based on the user's expectations. These expectations are often subjective and difficult to formalize.

IV. Integration of ECUs using CAN networks

In this section we compare the integration process of ECUs in bus-based CAN systems with star couplers, especially the CAN router. We then discuss the influence of the router on the prevailing development model in the automotive domain.

A. Integration of ECUs using CAN networks

As of today a typical car consists of several CAN buses with often different bandwidths. A typical setup consists of ECUs that send sporadic and periodic messages. It is also common that messages are specified in a way that the message should be sent as soon as an event occurs, but at least once within a given period.

As mentioned earlier, different subsystems need to share information with each other (e.g., powertrain and body). A CAN gateway is a device that bridges buses. Depending on the implementation, gateways provide a range of functionality like CAN identifier filtering, which is important if high and low speed buses are connected, identifier translation and repacking of message data (e.g., combining data of several messages into a new one).

The limited composability and compositionality of CAN-based systems in today's cars makes integration of ECUs a challenging task. As emerging behavior and side effects might influence the system, integration can only start if all components are available and tested. A delay of one component can therefore influence the whole development cycle of the project. Integration of ECUs influences the timing of messages on the bus. It is obvious that higher priority messages influence the timing of lower priority messages, but also lower priority messages have an influence as CAN messages are non-preemptive. To cope with this situation, CAN buses are traditionally only utilized up to 60%.

B. Star couplers

As traditional bus-based CAN has severe dependability limitations like missing fault isolation for babbling idiot failures [23] or inconsistent message duplication [12], star coupling devices were introduced. All of them share the design

decision to split the bus into distinct CAN segments. These segments usually consist of one or more ECUs. Segments are then connected to a central hub (i.e., the star coupler). While star couplers provide partitioning into segments, the amount of services varies depending on the concrete implementation.

There are star couplers that solely focus on improved fault detection and isolation as for example CANCentrate [3], which guarantees in-bit response time but still broadcasts every message from every ECU that has not been detected as faulty to every other ECU.

On the other end of the spectrum there is the time-triggered CAN router [17]. A router port receives messages from its connected bus segment, processes these messages and selectively forwards them to one or more destination segments. The router implements a star topology and has a strong focus on fault detection, while also providing extended higher level services [13]. These include *message rate control* like monitoring and enforcing minimum and maximum interarrival times, *message multicasting* which allows selective multicasting of messages instead of broadcasting every message to every other ECU, *identifier validation and translation* and *message checks and content translation*. In the context of this paper the capabilities of monitoring and enforcing maximum and minimum interarrival times are important. If a message violates the specified minimum interarrival time the router blocks that message and logs the violation. With the help of that service a faulty ECU can only have a limited influence on other CAN segments.

Another important service in the paper's context is the service of multicasting. The router does not broadcast every message to every other segment, it selectively forwards the message to ECUs interested in that particular message. On one hand this allows to use the existing bandwidth more efficiently and on the other hand this separation increases composability. We realized these higher level services by implementing a store and forward behavior. For a complete overview of the router's services, please refer to [17], [13].

C. Impact of the CAN Router on the Development Process

The main influence on the design phase is that the system designer *can* – but is not obliged to – make use of the router's services. It is of course possible to pick a subset of the services that fits the requirements best. The system designer can partition the bus into segments at a finer granularity than performed with today's central gateways. Thereby, the foundation for fine-grained fault isolation (e.g., message checks in the temporal and value domain) between ECUs and better utilization of the communication resources is established. Factors influencing the partitioning include communication relations between ECUs and the desired level of fault tolerance. For example, if a set of ECUs communicates only among themselves, they might be placed on the same CAN segment. If one of these ECUs also provides critical data for another segment, it might be desirable to use an entire CAN segment just for this ECU. By separating this ECU and making use of the router's services (e.g., message rate control, identifier validation), the system designer can limit the influence other ECUs like babbling idiots can have on this critical one.

Additional considerations in the design phase highly depend on the level of services the designer wants to make use of. If the designer wants to make use of the enhanced fault isolation capabilities he/she can specify minimum interarrival times for messages. If he/she wants to make use of the message check service, the designer can provide check functions that are then executed with the data of a message as a parameter (e.g., range checks).

In the implementation phase the specification that implementers of ECUs get does not substantially differ compared to a bus-based network. It is an advantage that router based setups specify the behavior in the temporal domain more accurately (e.g., minimum and maximum interarrival times).

Testing is simplified because the support of compositionality of the router basically allows to test the subsystems individually. The degree of independence depends on the interaction of these subsystems. If subsystems interact with each other, they form a "virtual" CAN bus. This virtual bus consists of all the CAN segments and ECUs that interact and has to be tested collectively like a single CAN bus. Additionally, we see the following advantages in the testing phase: The router continuously checks the timing assertions per message without any overhead in the ECU. Enforcing minimum interarrival times also provides the baseline for bounded message latencies, which then can be used as the foundation for the validation process. The router logs every violation reported by a port. These logs contain the type of violation, additional data like the CAN identifier and the source port number and a time stamp. This allows to build a chain of causality without the need of establishing a global time base in the CAN system itself.

Overall, system integration is simplified compared to standard bus-based CAN where existing services are invalidated (i.e., compositionality) or lead to unmeant effects on the system level (i.e., composability).

V. Evaluation

For evaluation we used an ACROSS [19] based prototype implementation of the time-triggered CAN router hosted on an Altera Stratix III FPGA development board. Our setup and the provided CAN traffic from Audi allow us to compare a traditional bus-based CAN setup with a setup based on the time-triggered router. The CAN traffic originates from a hybrid car's gearbox which uses five ECUs in this subsystem. In this section we first describe our test setup and continue to present the test cases we used for the evaluation. We conclude with the gathered test results.

A. Test Setup

We use two different setups: One traditional bus-based CAN setup and one CAN router based setup. Both setups are hosted on an FPGA and model the whole distributed system consisting of ECUs with CAN controllers, CAN buses, and especially the CAN router.

Figure 2 provides an overview of the router setup. In the following we describe the role of every component:

1) ECUs: We use NIOS II soft-core CPUs for emulating in-car ECUs. These ECUs are also hosted on the FPGA and are equipped with a CAN controller. We use these emulated ECUs to generate CAN traffic according to the car manufacturer's specification. This specification includes for every CAN

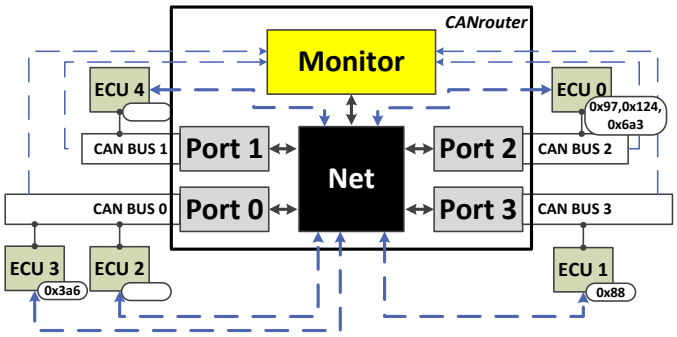


Fig. 2: Router test setup

CANID (period in ms)	Producer	Consumer
0x97 (10)	0	1,2,3
0x124 (20), 0x3aa (100), **0x6a3 (500)**	0	2,3
0x6b2 (1000), 0x5f0 (200), **0x88 (10)**, 0x30b (50), 0x6c2 (200)	1	4
0xa5 (10), 0xa6 (10)	1	3
0x9c (10), 0x3af (100), 0x3ac (100), 0x6a4 (500)	2	3
0x3ab (100)	2	0,3
0x98 (10), 0x99 (10)	2	1,3
0xa3 (10)	3	1
0xa4 (10)	3	0,1,2
0x6a2 (500), **0x3a6 (100)**	3	0,2
0xa3 (10)	3	1
0x3a7 (100), 0x90 (10), 0x91 (10), 0x92 (10), 0x3a8 (100), 0x123 (20)	3	2
0x6d1 (200), 0x89 (10), 0x568 (500)	4	1

TABLE I: Gearbox subsystem traffic

Identifier (ID) the sending and receiving ECUs, a message period and for a subset of the messages an inhibition time (i.e., minimum interarrival time). Table I summarizes the provided traffic. A ECU stores a time stamp in the data field of the message as soon as a message transmission is requested. If the message is delayed due to arbitration we do not alter the time stamp stored in the message. This time stamp is then used for calculating the end-to-end delays. To simulate the asynchronous timing of a CAN system, every ECU uses a different local clock offset that is taken into account when sending messages. If an ECU wants to send multiple messages in the same period, or one period is a multiple of another period (e.g., 10 and 100), the ECU forwards these messages to the send buffer of its CAN controller in the order of their priorities. The Commercial of the Shelf (COTS) CAN controller Intelectual Property (IP) core used in our evaluation implements a First In, First Out (FIFO) queue. This may lead to head of line blocking [6] of messages in the controller as it would occur in real-world systems.

2) CAN Router: The router contains 4 active and independent ports connected by an interconnect (i.e., the black "Net" block in Figure 2). The configuration of the interconnect allows every port to communicate with every other port. Every port contains a configuration that specifies message properties like minimum and maximum interarrival times, the set of valid message IDs and the destination ports. Additionally, every port contains a COTS CAN controller IP core.

3) Monitoring Unit (MU): For the bus-based setup we added one additional node. The MU, denoted as "Monitor" in Figure 2, reads every message received on any CAN bus and logs the delay. The delay is calculated from the intended send point stored by the ECU in the data field of the CAN message and the actual point in time when the message is received by the MU. As the whole setup is hosted on the FPGA, the sending ECU and the MU share a common clock signal. This allows us to measure delays precisely because local clock offsets, besides the known and intended offsets described in Section V-A1, are avoided.

For the CAN router based setup we added additional CAN controllers to the MU in order to monitor all CAN buses that are connected to router ports. As soon as the MU monitors a message, it logs the delay of that message. A message sent from one bus to another one has therefore two log entries. The first one contains the delay from the intended send point until it was successfully sent on the source bus and a second one when it was successfully sent on the destination bus (end-to-end delay).

4) CAN Buses: All CAN buses are configured at a speed of 250 kbit/s.

B. Test Cases and Results

For both test cases, traditional CAN bus, as well as router setup, we used the traffic shown in Table I. It is important to note that *all* of the messages shown in the table are processed in the bus setup as well as in the router setup. For every test case we show two results. The first one shows the mean delay of messages with a sampling period of $250ms$. Due to the high number of messages we chose to visualize a subset of messages that is in bold face in Table I. We chose five candidates out of the five fastest message periods, with a preference of the fastest period where two messages have been selected. This allows us to reason about the delays of selected messages in the average case. The second result we present shows the minimum and maximum delays for every message with a given ID. This fine grained view allows us to reason about the best and worst case message delays.

1) Test Case 1 (TC1) - Bus based CAN: For the first test case we configured the test environment in such a way that all ECUs are connected to a single CAN bus. In addition to the ECUs we connected the MU to the bus for monitoring purposes. The first test case allows us to study the behavior as we expect it to occur in a real car, thereby providing the baseline for the comparison with the router. Figure 3 shows the gathered averaged results over time, Figure 4 shows the minimum and maximum delays of messages on the bus.

2) Test Case 2 (TC2) - Four port CAN router: For the second test case we realized four CAN segments, where three segments contain one ECU and one segment contains two ECUs. The allocation of ECUs to buses and therefore to router ports is based on the traffic from Table I. We realized one CAN segment that consists of multiple ECUs (i.e., ECU 2 and ECU 3) to study the mutual influence on message latencies of ECUs sharing one CAN bus. We chose these specific ECUs because most of the messages are sent between these two ECUs. Separated from these two ECUs we placed ECU 0, which produces three of the monitored messages with a short period and also communicates to three other ECUs on a distinct CAN segment. This allows us to reason about the

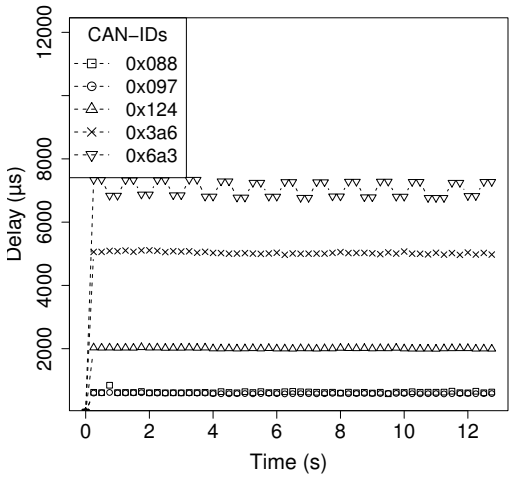


Fig. 3: TC1 (Bus based) - Averaged delays over time

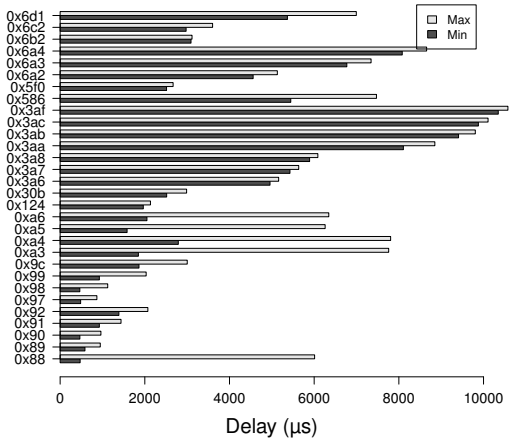
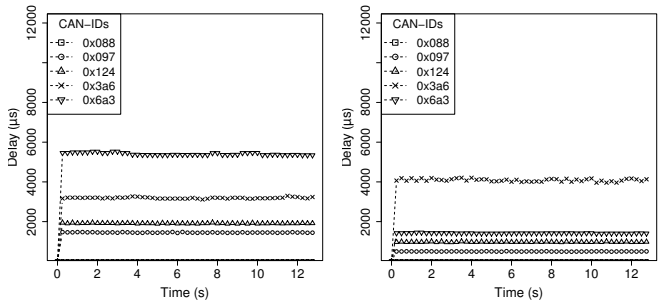


Fig. 4: T1 (Bus based) - Minimum and maximum delays

influence of the router. ECU 1 and ECU 4 exchange most of their messages with each other and are therefore placed on distinct CAN segments separated from the other ECUs. Figure 5 shows the averaged results for the most interesting two buses. Messages sent on bus 1 and 3 did not show a new maximum of the overall worst case delay. We intentionally omitted these two buses due to the limited paper length. Figures 6 and 7 show the monitored minimum and maximum delays.



(a) Bus 0 (b) Bus 2

Fig. 5: TC2 (Router based) - Averaged delays over time

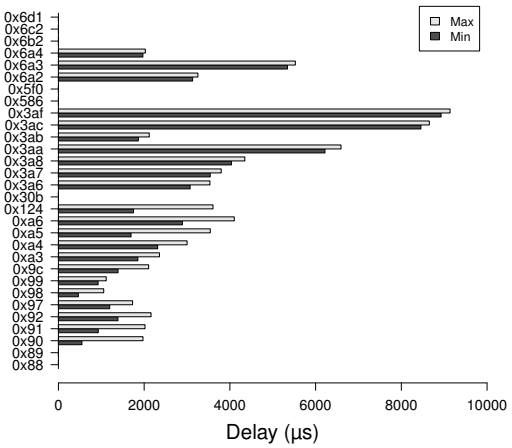


Fig. 6: TC2 (Router, bus 0) - Minimum and maximum delays

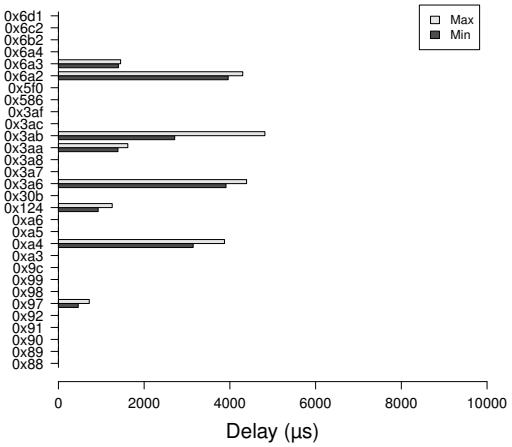


Fig. 7: TC2 (Router, bus 2) - Minimum and maximum delays

VI. Discussion

The first test case provides the baseline for the comparison with the router setup. In Figure 3 we see a CAN bus behavior as it would expected by the priority-based arbitration of CAN [22]. The three messages we selected (i.e., $0x124$, $0x3a6$ and $0x6a3$) show a delay corresponding to their priority in the average case. In Figure 4 we see a high jitter even for messages with a high priority. Reasons for that include the non-preemptive nature of CAN and head of line blocking in the ECUs themselves.

The router setup on bus 0 shown in Figure 5a provides some interesting insights. First we see that the delays of messages are still ordered according to the message priority. Even though messages $0x97$, $0x124$ and $0x6a3$ originate from a distinct bus segment (i.e., bus 2) the router successfully delivers these messages to their destination bus 0 where the destination port that finally sends the messages out on the bus has to compete with messages sent from ECUs 2 and 3. High priority messages delay messages with the ID $0x3a6$, but messages with the ID $0x3a6$ win the arbitration against the lowest priority message ID monitored ($0x6a3$). The most interesting observation is that delays for the average case and for the worst case are decreased. Compared with the bus setup, the delay of the messages with the ID $0x6a3$ decreased from a maximum of $7344\mu s$ to a maximum delay of $5344\mu s$. For messages with the IDs $0x3a6$ it decreased from $5161\mu s$ to $3535\mu s$. As the router

implements a store and forward behavior and the messages with the ID $0x97$ are produced on bus 2, it follows that we see a router introduced increase [13] of the end-to-end delay of these messages in the best case as well as in the worst case.

Bus 2 (Figure 5b) at first shows an unexpected behavior, as the delays of the messages are not ordered according to their priority any more. The message with the ID $0x3a6$ already has an average delay of about $3192\mu s$ before it is successfully sent on its source bus 0 (Figure 5a). Then it takes the router introduced delay until it reaches its destination port where it then again has to compete with the messages sent on destination bus 2. A message has to compete with other messages twice, first on the source port and second on the destination port. Only after the message is then successfully sent on bus 2 we can monitor the final end-to-end delay, which is in the average case $4089\mu s$. This is still below the one monitored in the bus-based setup (on average $5030\mu s$).

Compared with the bus-based setup the end-to-end delay of high priority messages is increased if they are sent from one port to another due to the router introduced delay. From Figures 4, 6 and 7 we deduce that the overall worst case message delay decreased from $10576\mu s$ to $9137\mu s$.

VII. Conclusion

In this paper we showed that bus-based CAN exhibits limitations with respect to composability and compositionality. We discussed the requirements for composability and gave an overview about the development model in the automotive domain. Then, we compared the prevailing integration strategy for bus-based topologies with an approach based on a time-triggered CAN router. We presented arguments how and why the CAN router supports composability. The evaluation with CAN traffic provided by a car manufacturer showed that the router is able to handle traffic from a real-world CAN system and that worst case delays got reduced. By means of separating CAN buses, enforcing minimum interarrival times and by selective multicasting the router provides the foundation for composability and compositionality. Separated segments do not interfere with each other if they do not communicate. If they communicate, the benefits on worst case end-to-end delays depends on the degree of interaction between ECUs placed on separated CAN segments. Due to the store and forward behavior of the CAN router, the best case delays of messages are increased. From the insights gained we conclude that a development process based on the time-triggered CAN router has the potential to significantly reduce the efforts undertaken by OEMs. We see the main advantage especially in the test, integration and validation phase of CAN based system development. Decreasing worst case delays at the cost of increasing best case delays makes the use of the router especially worth considering domains where high dependability requirements are not met by traditional bus-based CAN.

Acknowledgments

This work has been supported in part by the European research project ACROSS under the funding ID ARTEMIS-2009-1-100208.

References

[1] ARTEMIS. Reference Designs and Architectures – Constraints and Requirements. Technical Report Report Version 13, ARTEMIS Architecture Group, 2006.

[2] Autosar development cooperation. http://www.autosar.org. Accessed: 2013-03-15.

[3] M. Barranco, G. Rodriguez-Navas, J. Proenza, and L. Almeida. CANcentrate: An active Star Topology for CAN networks. In *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, pages 219 – 228, sept. 2004.

[4] J. Berwanger, W. Kuffner, M. Peteratzinger, G. Reichart, and A. Schedl. FlexRay - Exploitation of a Standard and Future Prospects. In *Proc. of Convergence 2006*, Detroit, MI, USA, October 2006. SAE.

[5] R.N. Charette. This Car Runs on Code. In *IEEE Spectrum*, 2010.

[6] Robert I Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Controller Area Network (CAN) Schedulability Analysis with FIFO Queues. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 45–56. IEEE, 2011.

[7] ETAS Group. http://www.etas.com. Accessed: 2013-03-15.

[8] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. *At NCOSE, Chattanooga, Tennessee*, 1991.

[9] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. AUTOSAR–A Worldwide Standard is on the Road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, 2009.

[10] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, Jean-Luc Maté, Kenji Nishikawa, and Thomas Scharnhorst. AUTomotive Open System ARchitecture-an industry-wide initiative to manage the complexity of emerging automotive E/E-architectures. *Convergence*, pages 18–20, 2004.

[11] Int. Standardization Organisation, ISO 11898. *Road vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*, 1993.

[12] J. Kaiser and M.A. Livani. Achieving Fault-Tolerant Ordered Broadcasts in CAN. In *Proc. of European Dependable Computing Conference*, pages 351–363, 1999.

[13] R. Kammerer, R. Obermaisser, and B. Froemel. A Router for the Containment of Timing and Value Failures in CAN. *EURASIP Journal on Embedded Systems*, 2012.

[14] H. Kopetz. A comparison of CAN and TTP/C. *Proc. of the IFAC Distributed Computer Systems Workshop, Como, Italy*, 1998.

[15] H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13:156–162, August 2002.

[16] T. Nolte, H. Hansson, and L.L. Bello. Automotive communications-past, current and future. In *Proc. of the 10th IEEE Conference on Emerging Technologies and Factory Automation*, 2005.

[17] Roman Obermaisser and Roland Kammerer. A Router for Improved Fault Isolation, Scalability and Diagnosis in CAN. *INDIN 2010*, Jul. 2010.

[18] OSEK VDX Portal. http://www.osek-vdx.org. Accessed: 2013-03-15.

[19] C.E. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek. The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 105–113, 2012.

[20] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering - Principles, Processes, Methods, and Tools*. SAE International, 2005.

[21] T. Steinbach, F. Korf, and T.C. Schmidt. Comparing time-triggered Ethernet with FlexRay: An evaluation of competing approaches to real-time for in-vehicle networks. In *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, pages 199 –202, may 2010.

[22] K. Tindell, A. Burns, and A. Wellings. Calculating Controller Area Network (CAN) Message Response Time. *Control Engineering Practice*, 3, 1995.

[23] K. Tindell and H. Hansson. Babbling idiots, the dual-priority protocol, and smart CAN controllers. In *Proceedings of the 1st Int. CAN Conference*, 1994.

[24] K.W. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS229, Dept. of Computer Science, University of York, June 1994.

[25] W. Voss. *A Comprehensive Guide to Controller Area Network*. Copperhill Media Corporation, 2005.

[26] Cornelia M Yoder and Marilyn L Schrag. Nassi-Shneiderman charts an alternative to flowcharts for design. In *ACM SIGMETRICS Performance Evaluation Review*, volume 7, pages 79–86. ACM, 1978.