A CAN-based Communication Model for Service-Oriented Driver Assistance Systems

Marco Wagner, Ansgar Meroth
Automotive Systems Engineering
Heilbronn University
Heilbronn, Germany
{marco.wagner,ansgar.meroth}@hs-heilbronn.de

Dieter Zoebel
Institute for Software Technology
University of Koblenz-Landau
Koblenz, Germany
zoebel@uni-koblenz.de

Abstract—**This paper describes a communication model for Service-oriented Driver Assistance Systems (DAS) basing on the Controller Area Network (CAN). In the field of DAS for truck and trailer combinations Service-orientation is a promising approach. One major demand to be able to deploy such systems in an automotive environment is the ability to allow service communication through CAN. This paper describes a CAN-based addressing scheme as well as a runtime adaption mechanism for Service-based DAS. The performance of the model is evaluated through experiments, showing that the approach is stable and showing good performance.**

Keywords - communication model; controller area network; CAN; service-oriented architecture; SOA; identifier assignment; driver assistance systems; automotive; in-vehicle communications

I. Introduction

State-of-the-art Driver Assistance Systems (DAS) like lane-keeping assistance or the electronic stability program are characterized by a static software and hardware architecture. This is because the number and topology of the Electronic Control Units (ECU) as well as the software modules forming the assistance do not change at runtime. In future, however, there will be DAS where the components of the system are distributed over more than one vehicle. These systems will have to be able to handle a high degree of distribution, changing hardware and software configurations and the integration of heterogeneous components at runtime. We call this class of systems Distributed Driver Assistance Systems (DDAS). One example are upcoming DAS basing on car-to-x communication. The main focus of our work lies on systems that support the driver while using a truck and trailer combination as the functionalities needed are distributed on both parts of the articulated vehicle. In these DDAS the driver may disconnect a trailer to connect another one carrying different sensors or software functionalities. Whenever the combination changes the system is challenged to achieve the best assistance possible using the functionalities provided by the truck and the trailer.

In [1] we presented a novel approach able to handle the demands of DDAS basing on Service-oriented Architecture (SOA). Hereby the functionalities are encapsulated into services. This allows distributing the components anywhere within one of the networks of the vehicle. Furthermore, services offer an abstract, well-defined interface to the outside world. Herewith, the heterogeneity of the different software functionalities is hidden. It also ensures compatibility between software components developed by different suppliers. In order to be able to react on changes at runtime software agents are used which re-configure the system by automatically discovering services currently available and re-orchestrating them to generate the overall functionality. The ideas on how such a re-orchestration process could be established are described in [2].

One major point in Service-oriented Architectures is the communication between the different services. In order to be able to use this paradigm in an automotive environment, a communication model has to be established to allow interaction between the components on the specialized automotive network systems. In the last years the Controller Area Network (CAN) has become the major communication medium used within cars. CAN is characterized by a message-oriented addressing scheme in the data link layer. All messages are sent in broadcast mode. As it is designed for embedded control systems the bandwidth of a maximum of 1 Mbit/s is rather small and the frame size is restricted to 8 Bytes of data. Even the latest extensions for CAN only allow a maximum of 64 Bytes of data using a flexible data rate [21]. Besides that, the identifiers used by the nodes are normally defined at design-time. In contrast, state-of-the-art SOA middleware systems rely on communication networks basing on the internet protocol (IP). These networks differ substantially from CAN as they make use of a node-based addressing scheme and interact mainly through singlecast messages. Furthermore they offer a significantly higher bandwidth and a much greater frame size. Finally, technologies like DHCP allow new nodes to enter the network at runtime. These circumstances set up several demands for a CAN-based communication model able to allow Service interaction:

(1) *Runtime adaption:* Identifiers have to be assigned to newly connected nodes automatically to allow changes at runtime.

(2) *Ensuring advantageous attributes:* The main characteristics of CAN, like message-oriented addressing, broadcasting and the absence of a central Network Master, should be preserved.

(3) *Interoperability:* It should be possible to run devices using the traditional, static approach as well as nodes using our newly developed model at the same time on the same bus.

(4) *Stability after initial configuration:* In the event of adding a new node to the bus, existing nodes should not have to re-configure their communication stack.

The remainder of this paper is organized as follows: Section II discusses several Service-oriented approaches for the automotive domain as well as techniques to deploy high layer protocols on CAN. In Section III our communication model is introduced. Section IV presents the results of our experiments while Section V calculates the worst case times of the algorithm. The last section concludes the paper giving an outlook on future work.

II. Related Work

A. Service-oriented approaches for the automotive domain

In recent years, there have been several approaches to apply SOA-principles to automotive software systems.

Shokry et al. in [3] are presenting an approach to use Service-based computing within the vehicle's software modules. The focus of their work is to manage software product lines in cars. However, they do not consider changes of the configuration at runtime and therefore developed no mechanisms to handle these changes. The approach presented by Krüger et al. in [4] also lacks of the ability to re-configure at runtime. Because of using a real time Common Object Request Broker Architecture (RT CORBA) middleware that runs on high level operating systems it also can't be ported to an embedded system that easily. Baresi et al. in [5] describe a system using an already existing SOA framework. Through basing on Java the system requirements are too high for being deployed on automotive ECUs. Furthermore, real-time constraints are not considered. These characteristics limit the field of application to the infotainment system of a vehicle. Besides that the proposal considers IP-based communication, only. The papers [6] by Eichhorn et al. and [7], written by Bohn et al., are describing systems basing on the Device Profile for Web Services (DPWS). DPWS also uses IP-based communication. Additionally [6] does not consider real-time conditions and only allows static, never changing configurations. The two approaches described by Xu and Yan in [8] and Ragavan et al. in [9] are focusing on a different scenario. In these systems not the internal functionalities are implemented as services but there are gateways that offer internal data of the car to the outside world. [8] is using this data to call Web Services located in the cloud, [9] on the other hand sets up an ECU using the Java-based Open Services Gateway initiative (OSGi) to allow a vehicle to invoke services offered by the outside world and vice versa. None of these approaches fulfill all the demands to be directly used within our context.

B. High-level protocols on Controller Area Network

As the previously mentioned approaches mostly depend on an IP-based communication one solution could be to run an IP-based protocol on CAN. This approach is adopted by Reichelt et al. in [10]. The basic idea is to assign every node in the network a single identifier. The IP addresses and ports of the sender and the receiver are moved into the data section of the frame. Since the size of an IP packet is normally much higher than the maximum frame size of CAN, all IP packets have to be split up and sent through several CAN messages. Although this approach is quite interesting, it has some drawbacks, too. First of all, the static assignment of identifiers to nodes does not allow changes of the CAN network at runtime. It also violates some of the main characteristics of CAN, as it changes the addressing mode from being message-oriented to being node-oriented. As the identifiers in CAN are also determining the priority of the message, the prioritization is no longer message-based but node-based. If now, for example, a node with a high priority identifier sends out a very long IP-based message, the bus may be blocked for a long time. As the addressing scheme of IP is maintained, the communication mode changes from being broadcast into being singlecast. Besides that the high level of fragmentation as well as the fact that the header and trailer of each packet are sent within in the data section of the CAN package causes a significant overload.

In order to avoid these drawbacks several researchers have published techniques to directly operate middleware technologies on the Controller Area Network. In [11] for example, Lankes, Jabs and Bernmerl present an approach to allow the usage of a CORBA middleware in CAN-based networks. A similar technology has been introduced by Kim et al. in [12]. However, both proposals assume a static assignment if identifiers to nodes within the network configured at design-time. Furthermore the maximum number of nodes is limited to 16 and 32 respectively. Another method is presented by Kaiser, Brudna and Mitidieri in [13]. The main idea is to deploy a real-time enabled middleware in Controller Area Networks. Again, the approach lacks of the opportunity to assign identifiers dynamically to the nodes but uses static ones instead. One well known technology for high-layer protocols on CAN is the DeviceNet standard [14]. In order to guarantee unique identifiers within the network, the identifier field partly consists of a combination of a vendor specific ID and the serial number of the device. This definition also leads to a static identifier assignment which is not sufficient within automotive SOA applications.

Other approaches are capable of handing out identifiers to the nodes connected at runtime. One of these is presented by Cavalieri in [15] to ensure real-time characteristics of a CAN network by dynamically changing identifiers and in doing so changing the priorities of the messages. Another approach is the standard CANopen [16] which includes a mechanism to change the identifiers of messages at runtime within the protocol stack. This mechanism is used and extended by Zhou et al. in [17] to build a self-organized protocol stack for networked control systems. However, all three approaches mentioned are basing on a central master device responsible for managing the identifier assignment within the network. Using such a master-based assignment requires maintaining an extensive database of nodes and their identifiers. Besides that a protocol has to be defined to keep this database consistent even in the event of disconnecting a node spontaneously by cuttingoff its power source. This scenario is a typical risk in embedded systems that has to be taken into account. Finally, the dependency on a central master device reduces the reliability of the network.

A last approach uses the reserved bits specified in the extended identifier specification (CAN 2.0B). In [18] Yellambalase and Choi introduce an automatic assignment mechanism operating without a central master device. The reserved bits of the extended frame are used as flags within a protocol to negotiate identifiers between the nodes of a network. However, the usage of reserved bits as well as the complicated negotiation mechanism requires the design of a specialized CAN controller and in doing so demands a disproportional effort.

III. A Communication Model for SOA in CAN Networks

None of the approaches mentioned in Section II is fully capable of running SOA-based systems on CAN with adequate performance. In order to achieve this, we developed a communication model consisting of an efficient and unique addressing scheme as well as a mechanism to dynamically add and remove nodes at runtime.

A. Addressing Scheme

One of the most significant attributes of Controller Area Networks is the fact that they are message-oriented. This means that other nodes are able to determine the content of the messages by reading the identifier. This principle allows establishing an efficient communication mechanism using broadcast messages. At the same time, the identifier defines the priority of the message and hereby of the data sent inside. In order to preserve these benefits the addressing structure of our communication model has to be structured carefully.

For the purpose of keeping the bus message-oriented, the identifier should allow to draw conclusions on the content of the message. This leads to a scheme including a so called Service Class Address (SCA). On the basis of the definition given by Rocco et al. in [19] we define a Service Class to refer to a specific functionality implemented as a service. In an automotive environment this may be, for example, functionality offering to retrieve the current steering angle. A Service Class Address is a number identifying a specific Service Class. A Service Instance on the other hand is an instantiated implementation of this Service Class. Using the SCA as an identifier would preserve the message-oriented communication style as the content of the message would be announced. However, this approach would not allow having several Services Instances of the same Service Class on the same network as the identifiers would not be unique anymore. This is why we combine the SCA with an Instance Identifier (IID). The IID is used to distinguish between several Service Instances of the same Service Class. The combined identifier ensures uniqueness and preserves the expressiveness stated in demand (2) in Section I.

Additionally the identifier is supplemented with the command that has to be executed by the service addressed.

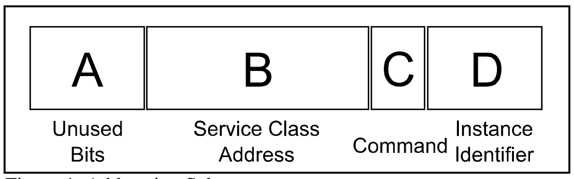


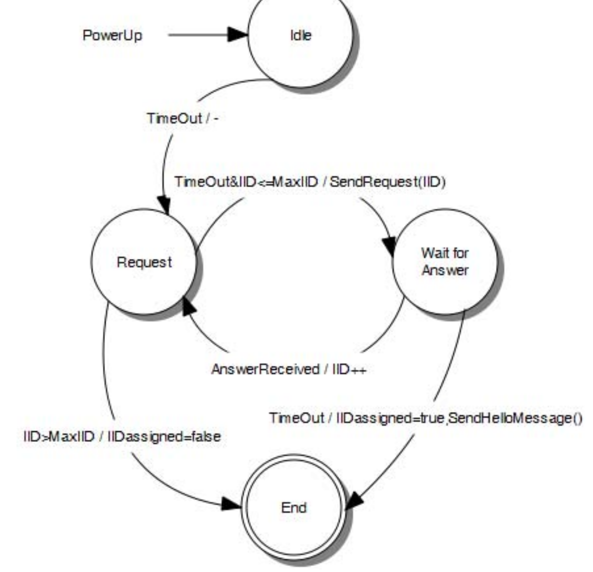
Figure 1. Addressing Scheme.

The overall composition of the identifier is shown in figure 1. The first bits of the identifier, that build section A, are not used within our addressing scheme. In doing so, the demand (3) to allow simultaneous deployment of services and classical functionalities on the same bus is ensured. This is because only one combination of these bits could be used for the service-based approach while all other combinations may be used for standard CAN messaging. Furthermore, these bits could also be used to set up virtual channels on the bus offering different priorities. In the automotive domain this could be used to, for example, define different channels for safety and comfort applications which, due to the broadcast-style messages, could still exchange data. The next section holds the SCA followed by the command bits included in section C. The least significant bits combined in section D are reserved for the IID. This order allows defining the priority of service messages and other messages on the bus, as the overall priority is mainly depending on section A. The order of the remaining sections ensures a prioritization primarily given by the functionality which is expressed by the SCA and hereby allows giving important functionalities preferential treatment. The influence of the command and IID fields on the prioritization is rather small. The actual size of the individual sections of the identifier may vary. A concrete example is given in section 4.

B. Automatic Identifier Assignment

The second part of the communication model is a mechanism to dynamically add or remove nodes at runtime. To avoid duplicate identifiers within the network it is essential to automatically assign them in the event of a system change. Our approach is based on the multi-master principle of CAN avoiding the usage of a central network master for administration. Therefore we developed a negotiation mechanism which enables the nodes to assign identifier among each other. As mentioned before, the identifier consists of three parts. The SCA describes the functionality offered and is hereby predefined for every service. The command bits are not describing the service but the action to fulfill and hereby can't be used to ensure uniqueness. This is why the identifier assignment procedure is basically an IID assignment procedure that makes sure that no two instances are using the same identifier.

The algorithm developed is presented in figure 2. It is a recursive, distributed algorithm basing on the Bully Algorithm used for electing a leader in a distributed system presented by Garcia-Molina in [20]. After powering up a device the service running on it is putting itself into idle state until a timeout occurs. The length of this timeout is chosen at random to avoid simultaneous bus access at startup of too many devices. After timeout the service reaches the Request state. In this state it waits until the bus is free. Again, it has to be made sure that no two services with the same SCA are accessing the bus at the same time. Therefore the node waits again for another random period of time. Then, it starts to send out messages to request an identifier. The message is send as a Remote Transmit Request containing no data bytes, carrying an identifier composed of the SCA, the command for a Service Request and the starting Instance Identifier which is 1. All other nodes within the network are now called to check this message. If there is a service in the network already assigned to this identifier it has to answer the request within a specified period of time. The requesting service holds the state "Waiting for Answer" until this period of time is elapsed or some other node has answered to this request. If there has been no answer, the algorithm finishes by assigning the request IID as the IID of the service. If there has been an answer to the request, the IID is increased by 1 and the process of sending out a request message starts again. The algorithm stops after exceeding the maximum Instance Identifier without assigning an identifier to the service and disables it.


PowerUp
Idle
TimeOut / -
TimeOut&IID<=MaxIID / SendRequest(IID)
Request
Wait for Answer
AnswerReceived / IID++
TimeOut / IIDassigned=true,SendHelloMessage()
IID>MaxIID / IIDassigned=false
End

Legend:

IID: Instance Identifier, initial value=1

IIDassigned: Flag to signal the result of the assignment process, initial value=false

MaxIID: Maximum Instance Identifier

SendRequest(): Method to send out a Request Message carrying the current IID.

SendHelloMessage(): Method to send out a Message signaling the identifier assigned.

Figure 2. Identifier Assignment Algorithm.

This negotiation approach fulfills the demand (1) set up in Section I of this paper as it allows automatically adding nodes by assigning identifiers to the services running on them. Furthermore the absence of a central network master as postulated in demand (2) is maintained. Finally demand (4) is also fulfilled as services, that already have an identifier assigned, do not have to be re-configured when a new node comes into the network.

IV. Experiments

In order to validate the functionality and the performance of the approach experiments have been run. For these experiments the extended identifier of CAN, containing 29 bits, has been used. Section A takes position on the bits 0 to 4 while section B is located between bit 5 and bit 20. The following section C consists of 3 bits and the remaining 5 bits are used by section D. This arrangement allows up to 65536 different Service Classes as well as 31 instances of the same class. The number of possible commands is set to 8 which are enough in our light-weight SOA approach. With these specifications experiments have been prepared to run six different scenarios. For all tests the following devices have been used:

- Embedded boards with an Atmel ATmega88 microcontroller running at 18.432 MHz as well as a Microchip MCP2515 CAN controller and a Philips PCA82C251 transceiver. The value for timeout when waiting for an answer to a request is set to 10ms.
- The automotive network testing and simulation environment CANoe 7.6.84 (SP4) from Vector. The value for timeout when waiting for an answer on a request message is set to 100ms.
- A CANcaseXL bus interface from Vector.

In the scenarios we distinguish between embedded ECUs running the algorithm in C on the ATmega88 boards and simulated ECUs being implemented in Vector's CAPL programming language running within the CANoe simulation. The assignment negotiation is run by Service Instances all belonging to the same Service Class. The experiments are logged using the trace function of CANoe. The data transmission rate of the network was set to 500 Kbits/s.

A. *Scenario 1*

The first scenario contains four embedded boards each of them running the identifier assignment algorithm. The boards were powered up simultaneously and start to obtain identifiers. This scenario simulates turning on the ignition when starting a car.

The experiment succeeded as all four participating nodes assigned themselves with a unique identifier. The time elapsed between the start of the process and the assignment of the last node was always less than 50ms.

B. *Scenario 2*

In the second scenario the same four embedded boards were used again. This time only three of the boards were powered up in the beginning. The fourth one was started after the other ones already successfully executed the algorithm. This scenario simulates adding a service at runtime into an already initialized network.

Again, the experiment went well. The three boards powered up at the beginning negotiated the identifier assignment in little more than 30ms. The fourth module ran successfully through the algorithm in up to 32 ms.

C. Scenario 3

This scenario combines the four embedded boards used in the other scenarios with three simulated ECUs to increase the number nodes involved. As in scenario 1 all nodes are powered up at the same time to simulate the start of a car.

Through to the longer timeout of the simulated ECUs it took about 430ms to assign identifier to all seven nodes. This time could be decreased significantly using lower timeouts in the nodes ran in CANoe.

D. Scenario 4

Again, a mixture of ECUs is used. Scenario 4 consists of four embedded boards as well as 4 simulated nodes. The embedded boards are powered up at the same time as three of the simulated ECUs. The fourth simulated ECU has already assigned an identifier at startup.

As the first IID is already in use the newly added nodes compete starting with the second IID. In comparison to scenario 3 the number of nodes increased which led to a longer assignment time. In this experiment it took about 470ms until the last service successfully obtained an identifier.

E. Scenario 5

This experiment represents a scenario were nodes are added on multiple events at runtime. One simulated ECU is already running at the start of the test. Three more simulated ones as well as three embedded boards are joining the network some time later. Finally, one last embedded ECU comes into the bus and requests an identifier.

The experiment ran well with timing characteristics comparable to the ones in the former scenarios.

F. Scenario 6

In scenario 6 all formerly used simulated and embedded ECUs are involved again. In order to validate that identifiers no longer used can be re-assigned again, the following sequence was executed: One simulated ECU was pre-assigned with an identifier. In a next step three embedded and three simulated ECUs were powered up. As soon as all identifiers are assigned one of the embedded ECUs is switched off. In a last step the remaining embedded board is powered up to obtain an identifier.

As a result the lastly connected ECU obtained the identifier originally assigned to the one that already left the network. The experiment proved that identifiers originally assigned to nodes that are no longer part of the network are automatically free to be used by newly added services.

V. Worst Case Times of the Algorithm

In the following chapter the algorithm is evaluated by calculating the worst case times. The scenario created is universal as all other scenarios can be derived from it. First of all, there might be nodes within the network that use fix identifiers. As the identifiers of these nodes can't be used anymore but might be requested by some node running the assignment algorithm each of these nodes causes a delay. This delay consists of the time needed for a Request Frame (t_{RF}), the Data Frame carrying the answer to the request (t_{DF}) as well as the maximum length of the timeout before sending another request (t_{TOR}). This delay occurs for every node with a fixed identifier and therefore is multiplied with the number of these nodes ($\#_{ENODES}$). Equation 1 presents the total delay caused by these nodes.

$$t_{ENODES} = \#_{ENODES} * (t_{RF} + t_{DF} + t_{TOR}) \quad (1)$$

Besides this delay, the assignment algorithm itself also needs some time to execute. The workflow of the algorithm is that a node waits for the timeout after the bus is free. In worst case this is the maximum timeout (t_{TOR}). After that, the node sends out a Request Frame which consumes t_{RF} to be proceeded. The whole sequence has to be repeated for every single node which assigns an identifiers ($\#_{NNODES}$). The overall amount of time for this sequence is calculated by equation 2.

$$t_{NNODES} = \#_{NNODES} * (t_{TOR} + t_{RF}) \quad (2)$$

To calculate the overall time elapsed until the assignment algorithm finishes t_{ENODES} and t_{NNODES} have to be summed up. Besides that, the maximum timeout in the beginning of the algorithm (t_{TOS}) as well as the maximum time that a node waits for an answer before it is allowed to use the requested identifier (t_{WA}) have to be added up. The overall time consumed by the assignment procedure in worst case is calculated as stated in equation 3.

$$t_{Overall} = t_{ENODES} + t_{NNODES} + t_{TOS} + t_{WA} \quad (3)$$

This universal scenario might be applied to all possible scenarios. In a normal startup scenario like the one simulated in scenario 1 of our experiments the number of existing nodes ($\#_{ENODES}$) is set to zero. The worst case times for scenario two could be calculated setting the number of existing nodes ($\#_{ENODES}$) to three and the number of new nodes ($\#_{NNODES}$) to one. These worst case times can now be calculated and compared to the times actually measured in our experiments. Scenario 4, for example, would calculate a total amount of 641.32ms. Since the total measured time averages to 470ms this is within the expectations. Calculating scenario 3 the ratio is 540.93ms calculated worst case time to 430ms measured time during the experiments.

VI. Conclusion and Future Work

By defining a CAN-based communication model we have shown that it is possible to run service-based systems on the most common automotive network. The model consisting of an addressing scheme and an identifier assignment algorithm has been tested using embedded hardware as well as simulated nodes in realistic scenarios. The tests have shown that the approach is working stable and with good performance.

Since now one of the fundamental parts of our SOA-based approach to handle DDAS has been developed, we can move on adding functionality to the services. Furthermore we plan to include other automotive network systems like FlexRay, Local Interconnect Network (Lin) or Media Oriented Systems Transport (MOST) to achieve a greater flexibility of our approach. The overall system will be implemented on a full scale prototype consisting of a car and at least two alternative trailers.

ACKNOWLEDGMENT

Marco Wagner has been supported by a grant of the "Thomas Gessmann-Stiftung", Essen, Germany.

REFERENCES

[1] M. Wagner, D. Zöbel and A. Meroth. "Towards an Adaptive Software and System Architecture for Driver Assistance Systems". *in Proc. of the 4th IEEE International Conference on Computer Science and Information Technology*, 2011, vol. 4 pp. 174-178.

[2] M. Wagner, D. Zöbel and A. Meroth: "Model-driven development of SOA-based Driver Assistance Systems". *in Proc. of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES '12) in conjunction with the IEEE / ACM CPSWeek '12*, April 2012, pp. 27-32.

[3] Shokry, H. and Babar, M.A., "Dynamic software product line architectures using service based computing for automotive systems," 2008.

[4] Krüger, I.H. and Gupta, D. and Mathew, R. and Moorthy, P. and Phillips, W. and Rittmann, S. and Ahluwalia, J., "Towards a process and tool-chain for service-oriented automotive software engineering". *in Proceedings of the Workshop on Software Engineering for Automotive Systems*, 2004.

[5] Baresi, L. and Ghezzi, C. and Miele, A. and Miraz, M. and Naggi, A. and Pacifici, F., "Hybrid service-oriented architectures: a case-study in the automotive domain". *in Proceedings of the 5th international workshop on Software engineering and middleware*, 2005, pp. 62-68.

[6] Eichhorn, M. and Pfannenstein, M. and Muhra, D. and Steinbach, E., "A SOA-based middleware concept for in-vehicle service discovery and device integration". *in Proceedings of the IEEE Intelligent Vehicles Symposium (IV) 2010*, 2010, pp. 663-669.

[7] Bohn, H. and Bobek, A. and Golatowski, F., "SIRENA-Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains". *in Proceedings of International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, 2006.

[8] Xu, Y. and Yan, J., "A Cloud Based Information Integration Platform for Smart Cars". *in Proceedings of the 2nd International Conference on Security-enriched Urban Computing and Smart Grids*, September 2011.

[9] Ragavan, S. Veera and Ponnambalam, S. G. and Ganapathy, Velappa and Teh, Joshua, "Services integration framework for vehicle telematics". In *Proceedings of the Third international conference on Intelligent robotics and applications*, 2010, pp. 636-648.

[10] Reichelt, T.; Oswald, N.; Windisch, A.; Forster, S.; Moser, H.; , "IP Based Transport Abstraction for Middleware Technologies". *in Proceedings of Third International Conference on Networking and Services (ICNS)*, Athens, 2007.

[11] Lankes, S.; Jabs, A.; Bernmerl, T.; , "Integration of a CAN-based connection-oriented communication model into Real-Time CORBA". In *Proceedings of International Parallel and Distributed Processing Symposium,* Nice, April 2003

[12] Kimoon Kim and Gwangil Geonand and Seongsoo Hongand and Sunil Kim and Taehyung Kim, "Resource-conscious customization of CORBA for CAN-based distributed embedded systems". In *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, Shenzen, 2000

[13] Jörg Kaiser, Cristiano Brudna, Carlos Mitidieri. "COSMIC: A real-time event-based middleware for the CAN-bus", *Journal of Systems and Software*, Volume 77, Issue 1, July 2005, Pages 27-36.

[14] The Open DeviceNet Vendor Association (ODVA). "DeviceNet – Technical Overview". White Paper. 2004.

[15] Cavalieri, S., "Meeting real-time constraints in CAN". In *IEEE Transactions on Industrial Informatics,* vol.1, no.2, pp. 124- 135, May 2005.

[16] CAN in Automation Consortium (CiA), "CAN Application Layer for industrial applications". Technical Specification. 1997.

[17] Zhou, Chun-Jie and Chen, Hui and Qin, Yuan-Qing and Shi, Yu-Feng and Yu, Guang-Can, "Self-organization of reconfigurable protocol stack for networked control systems". *International Journal of Automation and Computing*, Volume 8, Issue 2, pp. 221-235, 2011.

[18] Yellambalase, Y. and Minsu Choi, "Automatic Node Discovery in CAN (Controller Area Network) Controllers using Reserved Identifier Bits". *In Proceedings of the IEEE Instrumentation and Measurement Technology Conference (IMTC),* , Paris, 2007.

[19] Rocco, D. and Caverlee J. and Liu L. and Critchlow T. J., "Domain-specific web service discovery with service class descriptions". *In Proceedings of the International Conference on Web Services*, Orlando, 2005.

[20] Garcia-Molina, H., "Elections in a Distributed Computing System". *Transactions on Computers, IEEE*, pp.48-59, Jan. 1982.

[21] CAN with flexible data rate, Specification v1.0, Apr. 2012.