Multi-Domain CAN Gateway with Monitoring Capabilities

Florin Cătălin Brăescu, Lavinia Ferariu, Corneliu Lazăr

Department of Automatic Control and Applied Informatics
"Gheorghe Asachi" Technical University of Iaşi, România
E-mail: cbraescu@ac.tuiasi.ro, lferaru@ac.tuaisi.ro, clazar@ac.tuiasi.ro

Abstract: The paper presents a multi-domain Controller Area Network (CAN) software gateway with monitoring capabilities, targeted to distributed systems with complex architectures. Besides allowing a large amount of data exchanged between numerous CAN units, the gateway uses the traffic monitoring in order to implement adaptive message routing. The messages are sent on the fastest route which is able to maintain adequate traffic balancing. For increased reliability in the management of the critical messages, the transmission is done based on the priorities deduced from message ID's. Besides, the gateway stores in a local memory the optimal routes corresponding to nominal conditions and recalculates the paths only if the traffic load changes on the involved CAN buses or any communication error occurs. The embedded gateway application is based on an OSEK compliant real time operating system and allows USB communication with a PC, in order to visualize extensive information about CAN communication. The real time performances of the suggested solution are demonstrated on a distributed system involving multiple interconnected CAN domains.

Keywords: embedded systems, gateway, message routing, multi-domain CAN system.

1. INTRODUCTION

The Controller Area Network (CAN) is one of the most employed communication technologies for handling critical data exchange. Its applications range from high speed networks to low-cost wiring of multiple devices, within domains like automotive, industrial automation or medical equipment. CAN provides event-driven communication on a single terminated twisted pair cable. Given its high speed, Ethernet is one of CAN's competitors. However, several features make CAN bus the preferred option for the distributed industrial systems requiring high reliability communication, with extensive error checking: the lower connection costs per node compared with Ethernet based networks, the higher network reliability due to the passive structure of the network, the ease of use, the high protection against electromagnetic interferences and the availability of CAN controllers within microcontrollers (Oertel, 2012). Other CAN's main competitors are the local interconnected network (LIN) and FlexRay (Eisele, 2012). Due to its low speed, LIN is mainly limited to communication with sensors and actuators. Usually, in contemporary cars, the operation of LIN sub-networks is still integrated by a CAN bus. On the contrary, FlexRay is recommended because of a bandwidth higher than the one guaranteed by CAN. However, the higher costs and the significant involved engineering effort determined a seldom utilization of FlexRay, only. Currently, almost all of FlexRay-based applications are reported in the automotive industry.

The CAN protocol characteristics allowed distributed systems' designers to integrate an increasing number of software functionalities, as well as a growing number of CAN nodes within a CAN domain. Automotive industry represents an illustrative example for the development of CAN bus applications. A large numbers of CAN devices are interconnected in nowadays cars - between 6 and 60 CAN interface instances exchanging thousands of signals (Eisele, 2012; Navet and Perrault, 2012). The CAN domains are targeting specific functionalities, like power-train system, multimedia system, body and chassis system or safety system. This leads to an obvious lack of bandwidth. For compatibility reasons, many efforts are constantly targeted to increasing the bandwidth and the data bit rate for systems based on the popular CAN protocol, as the specifications of this protocol are still fulfilling the other requirements of engineering applications. The straightest solution for reaching this goal is allowing an appropriate device clustering, each cluster representing a CAN domain in a distributed system with multiple CAN domains. The interconnection between CAN domains would be made by means of gateway devices in charge with passing the data received from one CAN domain to the other CAN domains.

Within this context, this paper presents a new CAN-CAN gateway device which allows integrating the CAN bus within distributed systems with a high number of devices and complex topologies (e.g., resulted by interconnecting CAN domains). The proposed gateway monitors the connected CAN buses and identifies the fastest message route. The information obtained during real-time operation about the traffic in each CAN domain permits assuring an appropriate traffic balancing between the CAN domains and allows the reconfiguration of the fastest route, whenever overloading conditions or other communication errors are detected. Aiming improved reliability in fulfilling the critical real time constraints, the CAN messages received by the gateway are

locally scheduled for transmission, based on the priorities associated over the CAN bus, instead of using the common First-In-First-Out (FIFO) mechanism. Besides, the embedded gateway application is developed in an OSEK compliant real time operating system which ensures increased safety and predictability. Please note that, although this original gateway solution is discussed for CAN based systems, the concept can be simply extended to other communication protocols, too.

The paper is organised as follows. Sections 2 presents the related research work regarding the increasing of CAN's bandwidth and data bit rate, the current approaches related to multi-domain CAN systems and the existing gateway solutions. The main characteristics of CAN protocol are discussed in Section 3, while Section 4 presents the basic features of the OSEK compliant real time operating system, namely ProOSEK. A detailed description of the proposed gateway algorithm is given in Section 5, with emphasis on gateway's new facilities. Section 6 presents the development of the suggested algorithm for ProOSEK

real time operating system and, afterwards, Section 7 demonstrates the capabilities of the gateway by means of several experimental investigations. Section 8 is devoted to conclusions.

2. RELATED WORK

The Controller Area Network is a mature technology (Bosch, 1991) with more than 20 years of use in many application fields. It is utilised in places where multiplexed and high-speed communications provided over a limited number of wires are needed. There are various approaches trying to make CAN more appropriate for nowadays distributed systems, where an increased number of CAN enabled devices and CAN messages require bigger bandwidth and data bit rate.

An effort for increasing the CAN bandwidth and the bus speed has been conducted during the last years. The CAN+ protocol is based on dual speed approach that employs two alternate bit rates (Sheikh and Short, 2009; Ziermann *et al.,* 2009) and permits up to 16 times higher data rates. The CAN+ based devices switch the bit rate, after the arbitration phase, from the lower to higher value, during the time slots, whenever classic CAN devices do not listen. Changing the classic bus topology from line one to star (or tree) topology is the solution proposed by (Kurachi *et al.,* 2010). This allows obtaining transmission speeds up to 10Mbps, by means of the Scalable CAN protocol. Mainly, Scalable CAN is based on a new ACK ("acknowledge") information field and employs a new collision resolution algorithm which guarantees the delivery of a message within a given time period. Another new alternative stays in CAN with flexible data-rate, which offers an up to 8 times bigger data field speed and a payload of 32 or 64 bytes. (Hartwich, 2012; Oertel, 2012). Anyway, this is a young, still developing technology, so other approaches based on the classic CAN protocol could be preferred in critical real-time systems. In this attempt, the gateway solution presented in this paper offers increased flexibility in connecting the domains based on standard CAN for any type of topologies, as basis for supporting significantly improved bandwidths. It permits exchanging an increased number of messages between an increased number of devices, within and between CAN domains interconnected in complex architectures.

Other approaches keep the CAN protocol and develop more complex distributed architectures by clustering the CAN enabled devices (Navet and Perrault, 2012; Braescu and Ferariu, 2012) and interconnecting the resulted CAN domains through gateways or bridges (Sommer *et al.,* 2006). Bridges were used in order to connect different CAN networks (Eltze, 1997) by sending the messages from one network to the other, followed by gateways, which applied more complex processing operations on the CAN messages. The gateways are used for interconnecting different protocol networks (CAN, FlexRay) (Schmidt *et al.,* 2010) or same protocol networks (CAN-CAN gateways) (Sommer and Blind, 2007; Seo *et al.,* 2008), which deal with different transfer rates and provide operations like message forwarding or message assembling. The presented gateway is designed for connecting CAN domains with different working speeds. Besides, message forwarding provides dynamic route selection, as well as traffic monitoring and balancing between CAN domains.

The main types of gateways have been proposed, namely the discrete channel one and the complex channel one. The discrete channel gateway (the software gateway) is composed by a set of discrete channel CAN controllers managed by a software application in charge with message handling. This type of gateway is flexible, but it requires a powerful processor to deal with real time operation. It is worth mentioning that in support of software gateways, microcontrollers with many communication controllers are already available (Lorenz, 2008). On the other side, the complex channel gateway consists in a CAN controller specifically tailored for the application, supporting a hardware-oriented development with no additional processor load. The advantages of discrete and complex channel gateways could be merged in modular gateways (Taube *et al.,* 2005). This paper presents a discrete channel gateway implemented in ProOSEK. In order to reduce the required processing load, the gateway stores the nominal optimal paths in a local memory and recalculates the optimal paths only if the nominal working conditions are not met.

Different algorithms have been proposed for processing the received messages and scheduling their transmission within the gateway devices. The Round Robin scheduler employed in (Sommer and Blind, 2007) serves the incoming messages within equally allocated frames, while weighted Round Robin or other priority-based scheduling algorithms can be employed in order to induce priorities between CAN domains or messages. In order to favor the transmission of critical messages, the suggested gateway algorithm uses a local priority based scheduling. The priorities are determined from the ID of the message. Besides, this scheduling is accompanied by the detection of faster communication paths and the avoidance of bus overloading.

The common architecture in nowadays car is the star one, all the CAN domains being connected to the same gateway. The

architecture is quite rigid and cannot deal with an increasing number of in-vehicle CAN domains and messages. The traffic can be affected by gateway bottlenecks and the number of CAN domains is limited to the number of CAN modules provided by the gateway. In this context, it is of great interest to find appropriate solutions for the design of more complex architectures, able to allow advanced and intelligent cooperation between the embedded entities. As explained later, the main benefit of the proposed solution stays in the fact that it provides significantly increased flexibility in designing the topology of real-time distributed systems with interconnected CAN domains. The resulted distributed system can ensure high bandwidth, as well and increased robustness, via the original mechanisms embedded within the gateway. These mechanisms are meant to message scheduling, traffic balancing, detection of communication errors, including detection of communication overload. In extension to the previous gateway algorithms presented in (Braescu *et al.,* 2011; Braescu and Ferariu, 2012), this approach implements a priority-based scheduling algorithm for sending the received messages and integrates an improved adaptive routing, which is able to balance the traffic between the CAN domains of the distributed system. As the suggested solution follows the discrete channel gateway concept, increased flexibility is obtained in terms of compatibility with applications involving different number of CAN adjacent modules.

3. CAN PROTOCOL

The proposed gateway device exploits the facilities of the OSEK compliant real-time operating system and CAN communication protocol in order to allow a better management of the CAN traffic with increased bandwidth, in distributed systems involving architectures with multiple interconnected CAN domains.

CAN is a protocol that provides multi-master broadcast serial bus communication, each device being able to send or to receive messages on the bus, but not at the same time. The messages are broadcasted and each CAN receiver decides to process or to ignore a certain message. One of the main advantages offered by the CAN protocol is the arbitration mechanism performed during message sending. This mechanism assures that if two messages are simultaneously sent on the bus and a collision is detected, the arbitration is won by the message with the highest priority and no data corruption or communication error appears. After an arbitration phase, the winning message is sent, while the devices that lost the bus arbitration will hold the transmission and retry to send their messages later. The arbitration mechanism assumes the uniqueness of each CAN message identifier on a certain bus. This allows identifying the message source and interpreting the meaning of communicated data. All the devices besides the transmitting one will receive the message and will use the identifier field in order to detect whether the message is relevant for them or not. In the unlikely case of messages with the same identifier field being sent simultaneously by two or more CAN devices on the same bus, the arbitration mechanism fails, leading to communication errors.

The CAN protocol provides an appropriate environment for an automatic detection of data transmission errors and for a centralized error management. The CAN protocol implements mechanisms of error detection, error handling and error limitation. The error detection one allows the detection of the following error types: bit error, frame error, acknowledge error, CRC error and bit stuffing error. The error handling mechanism is based on the error flag field to signal the error's detection, as well as the error indication receiving confirmation, whilst the error delimiter field permits the bus nodes to restart communications after an error has occurred. The error limitation mechanism allows the CAN bus to work correctly even if there are errors-generating faulty nodes. Therefore, each node can be in one of the following three error states: error active, error passive and bus off (the node can neither send nor receive).

The message transmitter sends the message again when an overload frame is received and the next transmission's start is delayed by the overload frame. The CAN protocol specifies that in order to delay the start of the next message, a node can generate maximum two consecutive overload frames.

The CAN protocol does not provide a deterministic response time for messages assigned with low priorities (Bril *et al.,* 2006), so, in order to assure a good utilization of the CAN bus, higher priorities should be assigned to the messages with short periods of occurrence (Davis *et al.,* 2007). The proposed gateway is able to deal with CAN domains with different working speeds. The transfer rate for each CAN domain is taken into account during the dynamic selection of each message route, in order to provide a fast transmission and traffic balancing. The maximum transfer rate on CAN bus is 1 Mbit/s, but the typical transfer rates employed within automotive and industrial environments are 500 Kbit/s, 250 Kbit/s and 125 Kbit/s.

4. ProOSEK REAL TIME OPERATING SYSTEM

In support of later discussion concerning the development of the OSEK-based gateway, a brief overview of the ProOSEK real time operating system characteristics is given in the following. ProOSEK is designed to support the development of safe and portable embedded applications (3SOFT, 2003). The ProOSEK's portability is assured by a set of standard objects and API services that make the OSEK-based applications compliant with a large range of hardware architectures.

All the ProOSEK objects are statically created and all their attributes must be set before running the application, the static configuration of all defined objects giving ProOSEK predictability and safety. ProOSEK offers flexibility and scalability as well, as a multitude of layouts results via different configurations of available objects. More precisely, all the ProOSEK objects used inside the embedded application have to be declared in the *.oil files. An *.oil file may include an implementation definition part (specifying the attributes supported by the current OSEK implementation and all their permitted values) and an application definition part (containing declaration statements for all objects used in the

C application with assignments of corresponding attributes and declaration of references to required connected objects).

As a real time operating system, ProOSEK provides diverse capabilities for developing multitasking embedded applications: quick commutation between different contexts, predictable interrupts handling, simple resource sharing mechanism, events' handling for synchronization operations, time management through counters and alarms, as well as centralized and local error management.

The concurrent processes within an ProOSEK-based multitasking application are the tasks and the interrupt service routines (ISRs). The tasks execution is supervised by the RTOS scheduler and they can be preemptive or non preemptive. ProOSEK is an event-driven real time operating system, so the scheduler might be activated in one of the following cases: the ready tasks queue is changed, the processor or a shared resource are freed, or there is an explicit call to the scheduler. The task priorities are fixed and the processor is allocated to the highest priority ready task. The tasks having the same priority are managed according to FIFO algorithm.

The ProOSEK processes (tasks or ISRs) may share passive resources, the ceiling priority protocol (PCP) being employed as a solution to prevent priority inversion and deadlock. PCP assures that during a critical sequence (while a resource is occupied), the resource owner process receives a higher temporary priority, equal to the highest priority of the task/ISR that may ask to access that resource, such that the process may not be preempted by another competitor entitled to use the same resource, until it does not release it.

One can notice that ProOSEK, like most of the RTOSs, offers special debugging facilities like an easy monitoring of stack usage, tasks execution and even CPU utilization, by means of OS Tracer, Stack and Run-time Checker, correspondingly.

5. ALGORITHM DESCRIPTION

The proposed gateway employs an original policy for the dynamic selection of fastest message routes. The routing algorithm advantageously exploits extensive information extracted by means of traffic monitoring performed on each CAN domain. The main functional blocks of the gateway are illustrated in Fig. 1. As detailed below, the gateway is in charge with message priority based scheduling and routing. The sequence in which the received messages are handled is given by the priorities locally deduced from the message ID. For each transmitted message, the gateway determines the fastest available route, based on the information obtained via traffic monitoring. The optimal route ensures the traffic balance within the distributed system. This implicitly involves the avoidance of overloaded buses.

5.1 Traffic monitoring

Once the message routing should ensure real time traffic balance, monitoring the traffic on each CAN domain becomes essential. Traffic monitoring is employed for each available CAN bus connected to the gateway, and includes the CAN module level and the gateway level. At CAN module level, the monitoring is done by extracting all the information regarding the messages received or transmitted on the corresponding CAN bus, like the frame type, the corresponding frame length and the number of allocated bytes.

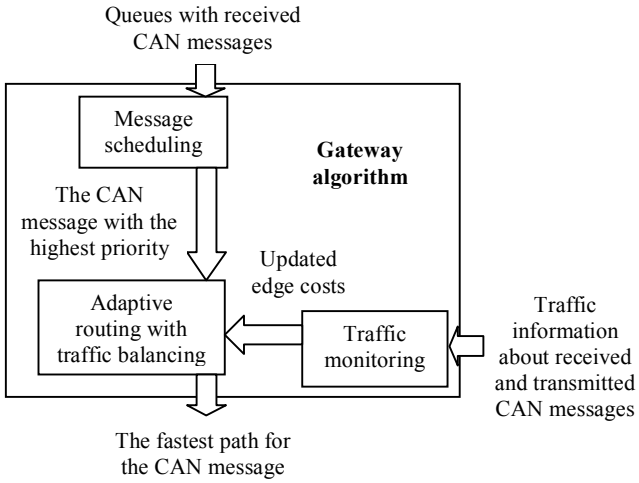

Fig. 1. The basic functionalities of the gateway algorithm.

The CAN messages can be of standard, extended, error, remote or overload frame. In the case of standard or extended frame, the CAN message is composed of the following fields: start-of-frame (SOF), arbitration, control, data, cyclic redundancy check (CRC), acknowledge (ACK) and end-of-frame (EOF). The arbitration field includes the message's identifier (ID) and the Remote Transmission Request (RTR) bit. More information about the contents of the CAN message, such as the number of data bytes available in the message and the frame's type (standard or extended) are encapsulated within the control field. The error frame is generated by any CAN node that detects a bus error. Once an error frame is received, all the CAN nodes drop the last received message and the transmitter resends the message. The error frame is composed by an error flag field and an error delimiter field.

A CAN node requests data from another CAN node by means of a remote frame, also called RTR frame. As a response to a RTR frame, a data frame with the same identifier will be transmitted from the CAN node which waits that RTR message. The nodes receiving a RTR frame check whether there is the corresponding transmitter defined or not. The node satisfying the condition sends the RTR response data frame. The request and the response frames are completely different frames, so it is obvious that the response frame could be delayed because of the messages with higher priorities exchanged on the CAN bus.

The monitor detects the type of data frame (standard or extended) and the length of data fields. The error, remote or overload frames are identified as well, and the error and overload information are recorded and used later by the adaptive routing with traffic balancing mechanism.

For a realistic estimation of the traffic on a CAN bus, the inter-frames spacing and the bit stuffing are considered. They are mechanisms offered by the CAN protocol in order to assure communication reliability. The first one assures that standard, extended and remote frames are separated by an inter-frame space, whilst the second mechanism ensures that a bit of opposite polarity is inserted after five consecutive bits of the same polarity. A good understanding of these mechanisms permits an appropriate traffic estimation to be made. The traffic estimation within the proposed gateway is done by considering both mechanisms. The length of the inter-frames spaces associated to standard, extended and remote frames are determined and recorded, yet the number of extra bits added by the bit stuffing mechanism is estimated. Exact information about the bits added by the bit stuffing mechanism can not be extracted, given the dynamic characteristic of the mechanism. One can notice that all these operations are done besides the normal message receiving operation.

The number of communicated bytes detected at CAN module level is saved for each CAN module into a local message queue. The traffic estimation at gateway level is done periodically by the task in charge with traffic monitoring that processes the local message queues contents and calculates the consumed bandwidth of each CAN bus. This is used further by the message scheduling and routing mechanism.

5.2 Message scheduling and routing

The message scheduling ensures that the message scheduled to be transmitted is the message with the highest priority from the CAN messages received by the gateway. This priority based policy replaces the common FIFO message scheduling, also employed in (Braescu *et al.,* 2011). The main benefit of the prioritized transmission lies in the fact that it can ensure the fulfilment of hard real time constraints by means of faster transmissions of critical messages. For the highest priority message, the routing mechanism decides afterwards which CAN module will be in charge with message transmission.

The CAN messages received by the gateway are locally scheduled for transmission based on the priorities associated over the CAN bus. The priority of a message is given by the identifier field which is 11 bits long (standard frame) or 29 bits long (extended frame); hence, the less is the ID's value, the higher is the priority of the message. For each CAN bus there is a set of received CAN messages that should be transmitted. Therefore, the gateway stores the incoming messages in multiple independent queues, each queue being associated to a distinct receiving CAN module. However, the scheduling policy acts on all these queues, meaning that the highest priority message is locally chosen from the whole collection of available messages. For a faster determination of the highest priority message, the receiving queues are preserved sorted in terms of the ID field, each message being inserted in the appropriate location. As consequence, the scheduling mechanism has only to check the first element of each queue and to select the message with the smallest ID for the next transmission. If two or more messages with the same priority are found, the message with the maximum predefined route cost will be preferred.

As presented in (Braescu *et al.,* 2011), each gateway locally stores the whole distributed architecture by means of a graph. The node of a graph corresponds to a communication node, while an edge indicates the existence of a communication bus. The cost c_i allocated for the edge i specifies the transfer rate of that CAN bus. The accepted values are $c_i \in \{1,2,4,8\}$, corresponding to the transfer rate 1000 Kbit/s, 500 Kbit/s, 250 Kbit/s and 125 Kbit/s, respectively. The routing procedure determines the fastest route in the graph by means of Dijkstra's algorithm.

In order to provide traffic balancing, multiple traffic costs are accepted for the same CAN bus. These costs are allocated online, by taking into account the consumed bandwidth reported for each CAN bus via traffic monitoring. For a specific bus, the algorithm considers a finite number of intermediary bandwidth levels and, correspondingly, a finite number of edge costs. More precisely, the edge cost is modified whenever the traffic on the bus exceeds or is below a certain threshold. The number of accepted bandwidth thresholds depend on the maximum bus speed; hence for a bus with a maximum 1000 Kbit/s transfer rate, 3 intermediary levels are considered (namely 500 Kbit/s, 750 Kbit/s and 875 Kbit/s), whilst for a bus with maximum 250 Kbit/s transfer rate, only one level is allocated.

The updated edge costs are further considered in order to identify the shortest route within the graph associated to the multi-domain CAN system. Once the shortest path problem is solved and the CAN message route is found, the message is sent to the chosen CAN bus. This adaptive routing assures traffic balancing over the whole distributed system and prevents bus overload situations. The risk of traffic overloads is significantly reduced due to the premature reconfiguration of the optimal communication paths, triggered by the gradual detection of traffic load's increasing. Although, given the fact that only a finite set of available costs is used, one can expect that the calls for the online reconfiguration of paths (by means of Dijkstra's algorithm) will rarely occur. Consequently, a reduced mean processing overload is anticipated to be introduced by the gateway.

A detected overloading situation is indicated by an overload frame generated by a CAN node either when a dominant bit is detected during inter-frames space or when a node is not yet ready to receive the next message. The last case could appear when the node is still reading a received message. The overload frame format is similar to an error frame format, but the overload frame can be generated only during an inter-frame space. If an overload frame is received on a CAN bus, the traffic monitoring mechanism will indicate the overloading situation by mean of the maximum value of the bandwidth corresponding to that bus. Then, the allocated edge cost will be increased, such that the bus will be ignored during the adaptive message routing, until the overload situation disappears.

6. OSEK-BASED GATEWAY DEVELOPMENT

The proposed gateway consists in a set of discrete channel CAN controllers offered by the microcontroller and an OSEK based software application in charge with message handling. The proposed embedded gateway application is developed in compliance with ProOsek implementation. For each CAN module, the gateway application uses an identical set of OSEK objects, in order to manage message reception and message transmission. For storing the received and the ready-to-be-transmitted CAN messages, two different message queues are employed, namely $RxMsgQueue_i$ and $TxMsgQueue_i$. By having separate queues for each CAN module, the resource sharing problems are avoided and better control on priority based insertion of received message could be easily provided.

The message transmission and reception mechanisms are solved by means of ISR objects. This allows achieving increased responsiveness of the application, as the priorities of ISRs are higher than the priorities of any task. However, please note that ISRs priorities are not managed by the real time operating system and the scheduler does not control their execution. Also, like in other real time operating systems, the API service calls allowed during ISR processes are restricted to a limited set.

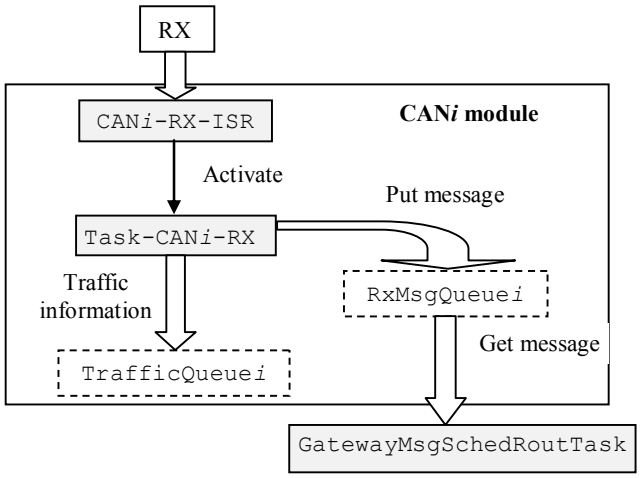

Fig. 2. ProOSEK gateway message reception mechanism.

The tasks of the application have been designed knowing that OSEK allows two types of tasks: basic and extended ones. A basic task can transit between ready, running and suspended states, whilst an extended task can have the waiting state as well (Lemieux, 2001). The extended tasks can have allocated their own events in order to implement synchronization operations, so an extended task can wait for an event to be set without occupying the processor. The extended tasks automatically switches from waiting state to ready state when the event is set (by another the task). It is worth mentioning that OSEK allows task's multiple activations, each task instance being executed as an individual process.

Fig. 2 illustrates the gateway reception mechanism. The $CAN_i\_RX\_ISR$ interrupt service routine is executed when a CAN message has been received in the CAN channel *i*. It activates the task $Task\_CAN_i\_RX$, which copies the message from the CAN reception buffer and saves it in the queue of received messages. As described in a previous section, the message is inserted in the $RxMsgQueue_i$ queue such that the messages are kept sorted subject to the ID field value, the first message in the queue being the one with the highest priority.

The $Task\_CAN_i\_RX$ task determines the received message frame type, estimates the number of received bytes and saves the computed value in a queue named *TrafficQueue*. This queue is locally managed, at CAN module level. The $Task\_CAN_i\_RX$ task is set as a basic, aperiodic, non preemptive task. It allows multiple activations, so each CAN message will be processed immediately after its reception, without disturbing the previous receptions, even the new reception overlays on the precedent ones.

The messages stored in the local queues associated to the gateway's CAN modules are examined by means of the *GatewayMsgSchedRoutTask* task. This task finds the message with the highest priority by checking the first elements from $RxMsgQueue_i$ queues, identifies its destination CAN domain, selects the appropriate route and saves the message in the ready-to-be-transmitted queue corresponding to the CAN module which will be in charge with its transmission.

The *GatewayMsgSchedRoutTask* task is basic, periodic, preemptive and it allows a single activation. The period of *GatewayMsgSchedRoutTask* is accurately managed by means of an alarm installed on the system counter object. Counter objects can be defined within OSEK application for providing high-level access to the existing hardware counters. Generally, the alarms associated to the counters can support a simple time management, by triggering the activation of specific tasks, the setting of certain events or the execution of associated callback functions. In this case, a periodic alarm is in charge with the activation of *GatewayMsgSchedRoutTask*.

The period of *GatewayMsgSchedRoutTask* task was set to be very small, such that the received messages are quickly processed. Besides, in order to avoid the loss of received messages, the priority of *GatewayMsgSchedRoutTask* is lower than the priority of $Task\_CAN_i\_RX$ task. This means that *Task_CANi_RX* is allowed to preempt *GatewayMsgSchedRoutTask*. As explained before, if two messages arrive at the same CAN module, at approximately the same moment, both messages will be processed promptly, because the $Task\_CAN_i\_RX$ task has multiple activations. However, please note that due to data consistency reasons, once *GatewayMsgSchedRoutTask* has chosen the highest priority message for transmission, the route selection continues with this message. All the newest incoming messages are treated at the next activations of *GatewayMsgSchedRoutTask,* even they have higher priorities than the message which has been already selected.

The gateway's message transmission mechanism is illustrated in Fig. 3. All the messages that need to be transmitted by a CAN_i module are saved in the corresponding $TxMsgQueue_i$ queue by the *GatewayMsgSchedRoutTask* task. The messages are read by the $Task\_CAN_i\_TX$ task, which is responsible

with finding an empty transmission buffer and storing the CAN message into it. Once a CAN message was sent on the bus, the end of the transmission is signalled by the corresponding transmission buffer by means of an interrupt serviced by *CAN$_i$_TX_ISR*, so that *Task_CAN$_i$_TX* can load another message ready to be transmitted. It is worth mentioning that setting appropriate dimensions of the queues involved within the gateway mechanism represents a key issue for preventing CAN message loss and for avoiding "bottleneck" effect. The *Task_CAN$_i$_TX* task is a basic, aperiodic, non preemptive task and accepts multiple activations.

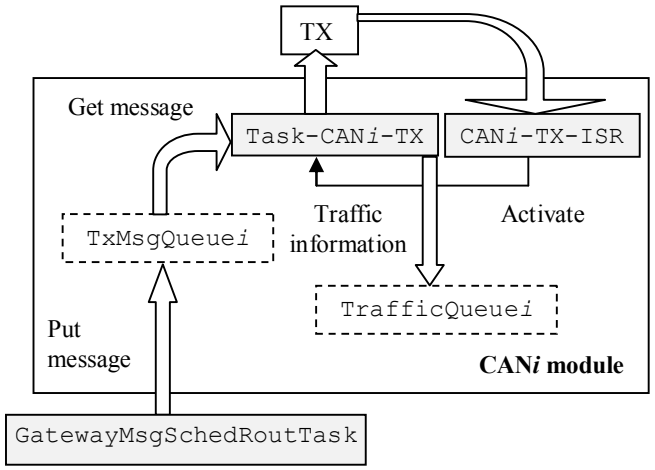

Fig. 3. ProOSEK gateway message transmission mechanism.

The ProOSEK-based solution proposed for traffic monitoring, message scheduling and routing is presented in Fig. 4. As mentioned before, the extracted traffic information regarding the received messages at each CAN module level is saved in the local *TrafficQueue$_i$* queue by the *Task_CAN$_i$_RX* task. It is further processed by the task in charge at gateway level with traffic monitoring, named *TrafficMonitorTask*. This task is a basic, periodic, non preemptive task, with a single allowed activation. It calculates the consumed bandwidth for each CAN bus and updates the edge costs within the system's graph representation, if situations of bus overloading or if the detected bandwidth is changed comparing to the thresholds.

Each edge cost change is broadcasted by the gateway to the other gateways by means of additional CAN messages assigned with the highest possible priority. These messages have the destination parameter set to each gateway domain existent within the distributed system and the data field specifies the CAN bus label and the updated edge cost. According to the common requirements of software gateways, the number of domains connected to a gateway should be kept reasonable low, hence the communication overload produced by these messages results acceptable. The *GatewayMsgSchedRoutTask* task recalculates the shortest route for a message only if a change occurred within the graph representation, otherwise it uses the already available information from the cache memory.

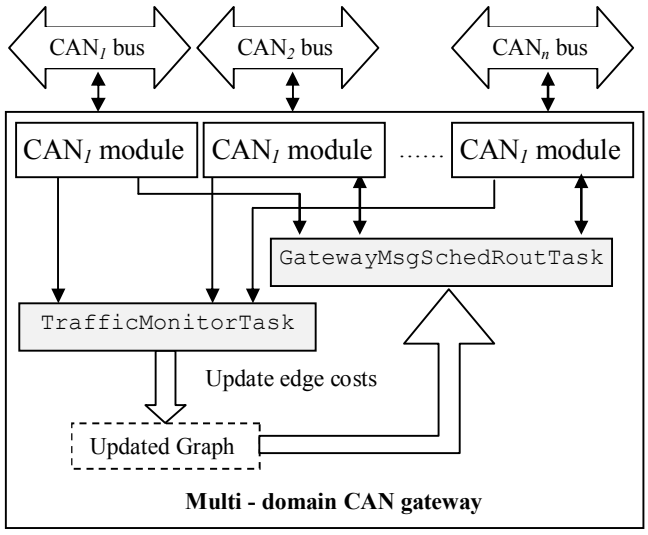

Fig. 4. ProOSEK gateway traffic monitoring and balancing mechanism.

7. EXPERIMENTAL RESULTS

The performance of the gateway mechanisms were experimentally verified on a laboratory setup that simulates a distributed system architecture with multiple CAN domains, as presented in Fig. 5. The system is composed of 8 CAN domains and 3 gateway domains. It can be treated as a distributed system from automotive industry, where its CAN domains are specifically allocated to systems like power-train system, multimedia system, body and chassis system, safety system or even to subsystems of them. The label attached on each edge indicates the cost (deduced from the bus transfer rate) and the corresponding CAN bus identifier.

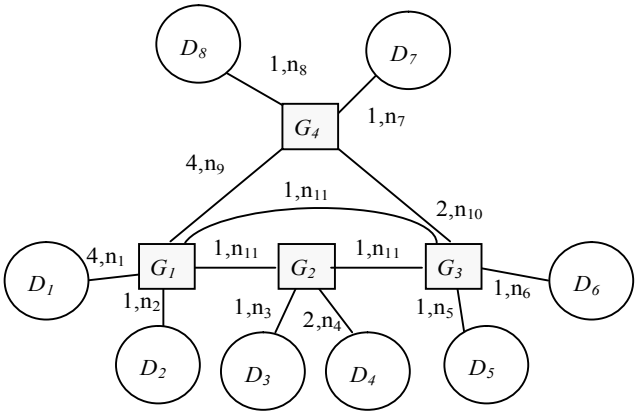

Fig. 5. Multiple CAN domain system architecture implemented during experiments.

The gateway development was tailored to the available hardware limitations. The hardware architecture considers a Freescale S12X microcontroller with 5 independent CAN modules. Each CAN module allows 1Mbps maximum transfer rate and offers 5 receiving buffers with FIFO storage scheme and 3 transmission buffers with internal prioritization. Given these specifications, the gateway can use

maximum 5 CAN connections. From software application development standpoint, an important feature of the S12X microcontroller is that the CAN modules have distinct interrupt vectors, which allows separate and customized CAN interrupt servicing by means of *CAN$_i$_RX_ISR* and *CAN$_i$_TX_ISR* ISRs.

Given the complex architecture of the multi-domain CAN system presented in Fig. 5, a large number of CAN devices clustered into 8 CAN domains were necessary in order to implement it. The experiments were done by using 4 gateway devices and 7 CAN monitor devices. The last ones are able to send different types of CAN messages and to monitor the CAN bus as well. Therefore, four of them (CAN Monitor 4, 5, 6 and 7) were used to simulate an entire CAN domain each (D_1, D_3, D_5 and D_7, correspondingly), by sending various messages at different periods. The experimental setup also considers the monitoring of the messages received by the simulated CAN bus. The other three CAN monitors (CAN Monitor 1, 2 and 3) were connected to the three gateways domains, in order to monitor the traffic on them. The embedded gateway application allows USB communication with a PC, in order to visualize extensive information about the CAN communication. The experimental setup structure is illustrated in Fig. 6.

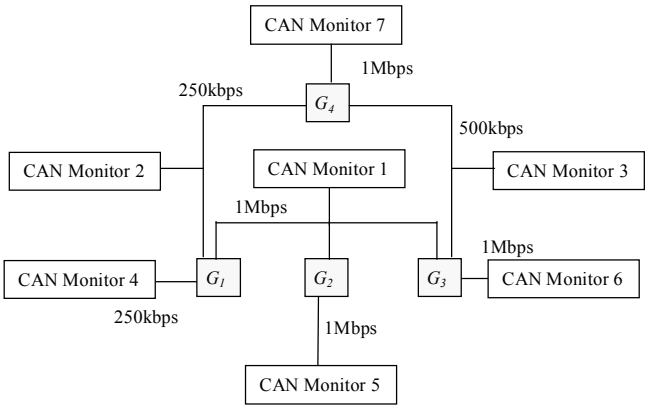

Fig. 6. Experimental setup for simulating a multi domain CAN system.

During the experiments, D_1 domain, as well as the n_9 and n_{11} buses are connected at the CAN1, CAN2 and CAN3 modules of G_1 gateway, D_3 domain and n_{11} bus are connected at the CAN1 and CAN2 modules of the G_2 gateway, correspondingly, and so on. Within each gateway algorithm, the *GatewayMsgSchedTask* task extracts the destination CAN domain identifier from the message data field and checks if the destination is connected at one of its CAN modules. If this condition is not satisfied, the task applies the Dijkstra's algorithm, in order to find the shortest path in the graph, which corresponds to the fastest route. Obviously, this path indicates towards which CAN module the received message CAN has to be forwarded.

For instance, a message sent from the D_1 domain to a device belonging to D_7 encrypts the identifier n_7 as its first byte of data, in order to indicate the final CAN destination. It was received by the G_1 gateway, which does not have the destination domain connected at one of its CAN modules and finally had to find the shortest (fastest) path in the graph. It determines the fastest route $G_1 - G_3 - G_4$ (of cost 3), instead of $G_1 - G_4$ (which has a cost of 4). During normal execution (without overloading), the message was sent on the bus corresponding to the $G_1 - G_3$ edge and G_3 sent it further to G_4. G_4 sent it further into D_7 domain, after identifying the CAN bus on which D_7 is connected. G_4 did not need to call the optimal routing algorithm, as the destination domain was directly connected to it. It is worth mentioning that the CAN message was not received by CAN Monitor 2, 5 or 6, confirming that the message was sent on the optimal route only.

A bus loading was implemented by increasing the number of messages sent by the CAN monitor connected to that bus. In order to test the proposed gateway monitoring capabilities, CAN Monitor 3 was used to send CAN messages within the same CAN domain until the occupied bandwidth value reached the 250kbps threshold value. When the traffic on n_{10} bus exceeded the threshold value, messages were sent by G_3 and G_4 in order to broadcast the updated edge cost. The messages were recorded by CAN Monitor 1 and 2. The messages sent from the D_1 domain to a device belonging to D_7 were received by G_1 that determined the fastest route at that moment, namely $G_1 - G_4$ (of cost 4), instead of $G_1 - G_3 - G_4$ (which has a cost of 5). When the traffic value on n_9 exceeded the 125kbps threshold value, another messages were sent by G_1 and G_4 with the bus label (n_9) and updated edge cost (8) encapsulated within the data frame. The same messages sent from the D_1 domain to a device belonging to D_7 and received by G_1 were transmitted on the fastest route at that moment, namely $G_1 - G_3 - G_4$ (of cost 5), instead of $G_1 - G_4$ (which has a cost of 8).

During the experimental tests, the communication overload produced by the transmission of the CAN messages announcing the graph updates was insignificant. These messages were sent by a gateway whenever its traffic monitoring module detected the need of changing the edge costs for any of the CAN buses connected to it.

8. CONCLUSIONS

The paper presents an improved OSEK-based gateway with new features and monitoring capabilities for multi-domain CAN distributed systems. It addresses the nowadays necessity for larger bandwidth and higher bit rate on the CAN bus by permitting distributed systems architectures with multiple CAN domains based on flexible topologies. The proposed solution makes use of traffic information on each CAN domain in order to assure adaptive routing of CAN messages, traffic balancing between CAN buses, as well as

local message prioritization. Message scheduling for transmission is employed at gateway level by using CAN messages ID field instead of using the common FIFO mechanism. The fastest routes are determined by using updated information about the bandwidth consumed on the connected CAN buses.

The embedded gateway application was developed in ProOSEK real time operating system, so it has increased predictability and safety. However the solution can be simply ported to other real time operating systems with event-driven scheduler. It also worth mentioning that even if the gateway algorithm and the related experiments are presented for distributed systems with multiple CAN domains, the algorithm itself is not limited to CAN protocol and it can be tailored for other communication protocols as well.

REFERENCES

Bosch, R., (1991). *CAN Specification Version 2.0*.

Braescu, C., Ferariu, L., Nacu, A. (2011). OSEK-based gateway algorithm for multi-domain CAN systems. *IEEE International Conference on Intelligent Computer Communication and Processing*, Cluj.

Braescu, C., and Ferariu, L. (2012). Multi-domain CAN gateway for complex manufacturing environments. *ModTech International Conference (Modern Technologies, Quality and Innovation - New face of TMCR)*, Sinaia.

Bril, R.J., Lukkien, J.J., Davis, R.I., Burns, A. (2006). Message response time analysis for ideal controller area network (CAN) refuted. In *Proceedings 5th International Workshop on Real-Time Networks (RTN'06)*.

Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J. (2007). Controller area network (CAN) schedulability analysis: refuted, revisited and revised. *Real-Time Systems*, Vol. 35, No. 3, pp. 239-272.

Eisele, H. (2012). CAN benefits in in-vehicle networking. In *Proceedings of the 13th international CAN Conference (iCC2012)*, Germany.

Eltze, J. (1997). Double-CAN controller as bridge for different CAN Networks. In *Proceedings of the 4th International CAN Conference*, Erlangen, Germany.

Hartwich, F. (2012). CAN with flexible data-rate. In *Proceedings of the 13th international CAN Conference (iCC2012)*, Germany.

Kurachi, R., Nishimura, M., Takada, H., Teshima, S., Mizashita, Y., Horihata, S., Yamamoto, H., and Natsume, A. (2010). Development of scalable CAN protocol. *SEI Technical Review No 71*, pp. 31-36.

Lemieux, J. (2001). *Programming in the OSEK/VDX environment*, CMP Books, Lawrence, USA.

Lorenz, T. (2008). *Advanced gateways in automotive applications*. Dissertation thesis, Germany.

Navet, N., and Perrault, H. (2012). CAN in automotive applications: a look forward. In *Proceedings of the 13th international CAN Conference (iCC2012)*, Germany.

Oertel, H.J. (2012). Using CAN with flexible data-rate in CANopen systems. In *Proceedings of the 13th international CAN Conference (iCC2012)*, Germany.

Schmidt, E., Alkan, M., Schmidt, K., Yuruklu, E., and Karakaya, U. (2010). Performance evaluation of FlexRay/CAN networks interconnected by a gateway. In *International Symposium on Industrial Embedded Systems (SIES)*, pp. 209-212.

Seo, S.H., Moon, T.Y., Kim, J.H., Kwon, K.H., Jeon, J.W., and Hwang, S.H. (2008). A fault-tolerant gateway for in-vehicle networks. In *IEEE Conference on Industrial Informatics*, pp. 1144–1148.

Sheikh, I., and Short, M. (2009). Improving information throughput in CAN networks: Implementing the dual-speed approach. In *8th International Workshop on Real-Time Networks (RTN'09)*, Dublin, Ireland.

Sommer, J., and Blind, R. (2007). Optimized resource dimensioning in an embedded CAN-CAN gateway. In *Proceedings of the IEEE Second International Symposium on Industrial Embedded Systems (SIES)*, Portugal.

Sommer, J., Burgstahler, L., and Feil, V. (2006). An analysis of automotive multi-domain CAN systems. In *Proceedings of the 12th EUNICE Open European Summer School*.

Taube, J., Hartwich, F., and Beikirch, H. (2005). Comparison of CAN gateway modules for automotive and industrial control applications. In *Proceedings of the 10th international CAN Conference (iCC2005)*, Italy.

Ziermann, S., Wildermann, T., and Teich, J. (2009). CAN+: A new backward compatible Controller Area Network (CAN) protocol with up to 16x higher data rates. In *Proceedings of DATE 2009*, IEEE Computer Society, Nice, France, pp. 1088-1093.

3SOFT (2003). *ProOSEK users guide*.