DDS on Top of FlexRay Vehicle Networks: Scheduling Analysis

RimBouhouch[1], HoudaJaouani[2], Amel Ben Ncira[3], Salem Hasnaoui[4]

Communications Systems Research Laboratory SYSTOM, Tunis, Tunisia

[1]rim.bouhouch@yahoo.fr; [2]jouani_houda@yahoo.fr ;[3]amel_benncira@yahoo.fr;[4]Salem.Hasnaoui@enit.rnu.tn

***Abstract*-FlexRay is a new communication system that offers reliable and real-time capable high-speed data transmission between electrical and mechatronic components to map current and future innovative functions into distributed systems within automotive applications. In the same context, the real-time middleware Data Distribution Service (DDS) is an appropriate alternative for the standard vehicular middleware considering that it handles Quality of Service (QoS) parameters. An interesting innovation would be the use of the DDS middleware on top of FlexRay networks formed by FlexRay cards to guarantee the arrival of the right data on the right time. In this paper, we will use as application an extended SAE(Society of Automotive Engineers) benchmark for the FlexRay network to identify the DDS DataReaders and DataWriters, and calculate the response time, based on the full scheduling model, and introduce it into the DDS QoS to further prove that the QoS of SAE benchmark are insured.**

Keywords- DDS; FlexRay; QoS; Scheduling; Response time

I. INTRODUCTION

FlexRay Networks are one of the newest x-by wire communication systems[1] known for their speed and performance which use ranges from planes to cars networks to insure communication over a shared medium. Thanks to its several features, this communication protocol is meeting safety critical applications performance requirements (flexibility, fault-tolerance, determinism, high-speed…). Therefore, FlexRay is emerging as a predominant protocol for in-vehicle x-by-wire applications (i.e. drive-by-wire, steer-by-wire, brake-by-wire, etc.). As a result, there has been a lot of recent interest in timing analysis techniques in order to provide bounds for the message communication times on FlexRay. On the other hand, the real-time Data Distribution Service [2] based on the subscription-publication paradigm offers a clear distinction between the communicating tasks by classifying them into DataReaders and DataWriters and that helps insuring the delivery of the right data on the right time. Vehicles real-time networks such as FlexRay are based on the interaction between the scheduler, reflecting the working process of the network, and the application in need of communication. But in this scheme a real-time middleware is necessary to guarantee better performances since most of vehicle manufacturers usually use AUTOSAR as middleware. The OMG (Object Management Group) Data Distribution Service provides a real-time Middleware that ensures the interaction between the physical layer and the application layer providing a communication pattern. The performances of this combination must be evaluated based on a real-time parameter to determine if the application requirements have been met.

Data Distribution Service is considered a standard in embedded systems implement. It is based on publish-subscribe communication model, and supports both messaging and data-object centric data models. DDS was designed for real-time systems; the API and Quality of Service (QoS) are chosen to balance predictable behavior and implementation efficiency/performance. One of the promising approaches is to make an efficient use of QoS mechanisms propose in the DDS specification when adopting real-time network such as FlexRay. DDS provides the DEADLINE QoS Policy, LATENCY_BUDGET Qos Policy, TRANSPORT_PRIORITY QoS Policy and other policies specifically targeted to minimum latency, predictable real-time operation in high-performance distributed data critical systems. However, DDS specification is less explicit about the scheduling mechanisms that should be used to coordinate these policies and to make the best benefit when exploiting the underlying facilities of the real-time network.

The Data Distribution Service (DDS) is an open standard managed by the Object Management Group (OMG) and representing the first general-purpose middleware standard that addresses challenging real-time requirements. In fact, DDS handles a wide range of data flows, from extremely high performance combat management or flight control to slower command sequences. This specification describes two levels of interfaces: A lower level Data-Centric Publish-Subscribe (DCPS) that is targeted towards the efficient delivery of the proper information to the proper recipients and an optional higher-level Data-Local Reconstruction Layer (DLRL), which allows for a simpler integration into the application layer. The data-centric publish-subscribe model employs a data-centric integration model to decouple applications composed of data providers and/or consumers spread onto different computers. Thus, it creates a global shared data-space that greatly simplifies integration. A data-object in data space, identified by its domain id, is uniquely identified by its keys and typed topic. The DCPS layer consists of the following entities: Domain Participant, DataWriter, DataReader, Publisher, Subscriber, and Topic. The principle is to transmit data directly from a publisher to all its subscribers with no intermediate servers. That is to say, applications communicate by publishing the data they produce and subscribing to the type of data they consume. They require no knowledge of each other, only of the data they exchange.

DOI:10.5963/IJCSAI0301002

DataWriter is a typed facade to a publisher; participants use DataWriters to communicate the value and changes to data of a given type. Once new data values have been communicated to the publisher, it is the Publisher's responsibility to determine when it is appropriate to issue the corresponding message. A Subscriber receives published data of different specified types and makes it available to the participants using a typed DataReader attached to the subscriber. The association of a DataWriter object with DataReader objects is done by means of the Topic.

DDS goes beyond simple publish-subscribe middleware and enables a "plug in" open architecture approach to integration. The key technology is the ability to capture all the timing, reliability, and other important interface constraints called "Quality of Service" (QoS) properties. Indeed, a Quality of Service is a set of characteristics that controls some aspects of the behavior of the DDS Service.

In addition to a set of QoS Policy values that are related to it, each DDS entity has a corresponding specialized listener object able to notify it of events. It can also be configured through QoS policies, be enabled and support conditions that can be waited upon by the application. Even if many other types of middleware support QoS, DDS still unique since it not only ensures reliability and timeliness thanks to its data-centric approach, but also automatically discovers matching publishers and subscribers, and then enforces the QoS contracts between them. This automatic discovery eliminates most configuration control issues and supports networks that change at runtime. With those facilities, DDS systems can add, modify, restart or update new modules without redesigning other interfaces and enables programs to evolve and mature without constant rework because system integration is done one component at a time without impacting other components.

We mention some QoSpolicies that are related to real-time parameters:

- The DEADLINE QoSPolicy expresses the maximum duration (deadline) within which a DataReader expects a data-object instance to be updated. If a sample is not received within the deadline, a listener method is called.
- The LATENCY_BUDGET QosPolicy provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.
- The LIFESPAN QosPolicy, on a DataWriter and Topic, which specifies how long the data written by a DataWriter is considered valid ("time to live").
- The TIME_BASED_FILTER QosPolicy specifies a minimum_separation value that allows a DataReader to specify that it interested only in (potentially) a sub-sampled set of the values for a data-object instance.
- TRANSPORT_PRIORITY QoSPolicy, in a DataWriter, which allows a DDS application to take advantage of transports that are capable of sending messages with different priorities.
- The RELIABILITY QosPolicy, on a DataWriter, DataReader, or a Topic. This policy determines whether a message should be sent best effort (send once without expecting acknowledgments) or reliably (resent until positively acknowledged).

With these QoS policies, and many others, the DDS publish-subscribe is suitable to the real-time communications context. In fact, DDS is adopted in hundreds of commercial and government programs for its several features. It also has growing footprint in commercial telecommunications, train, automotive, medical, science and financial applications.However, there is not enough works dealing with implementing DDS upon real-time networks except those addressing the Controller Area Network (CAN) [3]. On the other hand, there is an active community of government, industry and academic participants using the OMG to clarify improve and update the DDS standard, ensuring its viability for years to come. Thirty-six companies are voting to adopt the original specification, including Borland, Ericsson, Fujitsu, IBM, Oracle, Real-Time Innovations (RTI), THALES, PrismTech, and Nokia. Ten companies now offer implementations of the OMG DDS specification, of which six offer commercial versions of the product. Some of the companies currently offering DDS implementations include: Real-Time Innovations [4] which has a product called NDDS that provides publish-subscribe architecture for time critical delivery of data. PrismTech[5] which has a product called Open Splice[6] compliant with real-time networking. Thales Naval [7] having a product called SPLICE [8] that provides a data-centric architecture for mission-critical applications.

In this paper, we propose the SAE benchmark as an automotive application to test our system based on FlexRay network formed by Fujitsu Cards MB91F465X and the DDS real-time middleware and evaluate its performances using the worst case response time calculated by the full scheduling model. The extended benchmark communicating tasks was classified according to DDS into DataReaders and DataWriters on each node of the network.

The remainder of this paper is organized as follows: In Section 2, we discuss the related work, comparing our approach with some existing solutions. Section 3 is dedicated to the extended SAE benchmark explaining the new signals and priority. In Section 4 we give a classification of the DataReaders and DataWriters according to the used benchmark. In Section 5 we summarize some parameters related to the scheduling in FlexRay networks. In Section 6, we give the equation relative to the full scheduling method and how to use it to determine the response time. The Section 7 is dedicated to the description of our studied architecture. In last two sections, we demonstrate how to approximate the DDS QoS using the response time and the obtained results and conclusions.

DOI:10.5963/IJCSAI0301002

II. RELATED WORK

A. SAE Benchmark

In 1993, SAE published a document on the signaling requirements for a prototype electric car with point-to-point communication Network [9]. Since then this document has been used as basis for building an intra-vehicle communication benchmark [10, 11]. The benchmark consists of six main modules in the vehicle's class C network: the vehicle controller, Inverter/Motor Controller, Transmission, Battery, Brakes, and the Driver interface and control panel. Fifty-three signals are shared among the different modules, some of them are periodic and some are sporadic. For sporadic signals, SAE does not provide any indication to the signal average period. Kopetz addressed this by assuming these times based on the application requirements [11]. Hence, He modeled sporadic signals as periodic ones with the period set implicitly to 20 msec. His argument was that the latency of periodic signals should be less than or equal to their period i.e. for 20 msec periodic signal the latency should be less than or equal to 20 msec. The 20 msec period was adopted based on the driver's response time to changes in the driving environment, and it hypnotizes that the driver can't sense latencies of 20 msec or less. Hence, Kopetz set the latency requirement for sporadic signals to 20 msec implying a period of 20 msec. However, Tindell and Burns [10] took a more relaxed assumption, and probably more realistic, by setting the latency requirement for sporadic signals to 20 msec, and the corresponding hypnotized period to 50 msec. A list of the all the fifty-three signals, their periods, and their latency requirement are captured from[10]. It is important to notice that the aforementioned signals were not designed for any specific protocol; rather, they represent the data that need to be exchanged between the five listed modules. Kopetz, designed his messages to best serve the TTP protocol when under evaluation, while Tindell and Burns organized the messages to best fit the CAN protocol. The later effort indicates that assigning each signal to a message will result in huge bandwidth consumption and will require using 250 Kbps bus speed at minimum, 500 Kbps preferably, in order to meet the latency requirement. Therefore, Tindel and Burns suggested combining multiple signals in a single message to reduce the effect of the protocol overhead and the contention delay [10]. Basically, sporadic signals of low rate were piggybacked in fewer messages of higher rate, called server messages. The resulting message structure contained 17 periodic messages [10].

B. Scheduling

Tasks in real-time networks such as FlexRay or CAN are scheduled according to a static or a dynamic scheduling method. A static scheduler is based on time triggered scheduling according to the Time Division Multiple Access (TDMA) technic where each participant is granted a specific fixed interval in a repetitive time window. TDMA scheduling guarantees a deterministic transfer of messages, but has a major disadvantage that the bandwidth is not used efficiently. A dynamic scheduling is an event triggered scheduling where participants can only send information if an event occurs, such as new data is ready for transmission.

Our previous researches [12] were interested in scheduling for the Data Distribution Service (DDS) architecture over CAN. We have developed in each node a local scheduling component, the Earliest Deadline First (EDF) scheduler. The latter, sends scheduling parameters of tasks to the global scheduling system. Then, information is sent to a distributed information collection service called the System Information Repository (SIR). In [13] we have presented how DDS API is implemented on top of FlexRay Driver. In [14], we have presented a combined scheduling method that can be applied for both static and dynamic scheduling in FlexRay.

Related studies to this research include time triggered, event triggered and automobile protocols.

First studies [15] illustrate how a window-based analysis technique can be used to find the Worst-Case Response time of a task. It considers sporadic activities, where tasks arrive sporadically but, then execute periodically for some bounded time.

The Paper [16] proposes an analytical framework for compositional performance analysis of a network of Electronic Controller Unit (ECU) that communicates via a FlexRay bus. The main contribution was a formal model of the protocol governing the DYN segment of FlexRay.

In this paper, we focus our interest on the scheduling on the FlexRay node and so propose a new scheduling method that handles all the delay sources to determine if the SAE benchmark QoS parameters has been met by calculating the worst case response time.

III. EXTENDED SAE BENCHMARK

In order to translate signals to messages, the numbers of nodes taking part in the network and the way they map into today's ECUs have been defined in [17].Then, signals can be assigned to the individual node queues according to their location and set of tasks. First, the SAE module names are translated to their equivalents in today's terminology. The Driver and Battery modules are combined in the Body Control Module (BCM), which acts usually as gate way between the low-data rate and the high-data rate networks in a vehicle. Similarly, the Vehicle Controller and Inverter/Motor Controller can be combined in a single node called Engine Control Module (ECM), while the Transmission Controller is denoted as the Transmission Control Module(TCM). Finally, the Brake controller in the SAE testbed Maps to the Hydraulic Brake Control Unite(HBCU), the Brakes Controller is usually called the Electronic Brake Control Module (EBCM) and the Steering Assist module is called

DOI:10.5963/IJCSAI0301002

Active Frame Steering(AFS), while the rest of the modules don't have common names, and hence keep their terminology unchanged. Table VIII summarizes all the signals in the extended benchmark and Table IX transforms those signals into messages and assign them relative priorities[17] are presented in the appendix.

IV. DATAREADERS AND DATAWRITERS CLASSIFICATION

In this paper we are interested in studying the QoS related to the communicating process, these communicating tasks are inspired from the SAE benchmark. In the DDS specification [2], communicating tasks are classified into DataWriters and DataReaders using Publishers and Subscribers to transmit and receive data. Per node, we can find one Publisher and/or one Subscriber attached to different DataWriters and DataReaders interested on different Topics.

Using the SAE benchmark we have classified the communicating applications into DataReaders and DataWriters and attached each one to the interested Topic in the vehicular network, as shown in Table I.

TABLE I.DATAREADERS AND DATAWRITERS CLASSIFICATION

Topic	ID	Packaged Signals		Transmitter		Receiver	
		N°	Signal Designation	Node	Data Writer	Node	Data Reader
Hi&Lo Contactor Open/Close	1	14	Hi&Lo Contactor Open/Close	**BODY Control Module**	HLC_W_TASK()	**Engine Controller Module**	HLC_R_TASK()
Accelerator Position	3	7	Accelerator Position		AP_W_TASK()		AP_R_TASK()
Brake Pedal Position	13	71	Brake Pedal Position		BPP_W_TASK()		BPP_R_TASK()
Acknowledgments	17	23	12V Power Ack Vehicle Control		ACK_W_TASK()		ACK_R_TASK()
		24	12V Power Ack Inverter				
		25	12V Power Ack I/M Control				
		28	Interlock				
Control Switches	18	15	Key Switch Run		CSW_W_TASK()		CSW_R_TASK()
		16	Key Switch Start				
		17	Accelerator Switch				
		19	Emergency Brake				
		20	Shift Lever (PRNDL)				
		22	Speed Control				
		26	Brake Mode (Parallel/Split)				
		27	SOC Reset				
Batteries Voltage and Current	31	1	Traction Battery Voltage		BVC_W_TASK()		BVC_R_TASK()
		2	Traction Battery Current				
		4	Auxiliary Battery Voltage				
		6	Auxiliary Battery Current				
Traction Battery Measurement	34	3	Traction Battery Temp,Average		TBM_W_TASK()		TBM_R_TASK()
		5	Traction Battery Temp,Max				
		13	Traction Battery Ground Fault				
Motor Speed and Torque	4	43	Torque Measured	**Engine Controller Module**	MST_W_TASK()		MST_W_TASK()
		49	Processed Motor Speed				
Pressure and Main Contactor	6	32	Clutch Pressure Control		PMC_W_TASK()	**BODY Control Module**	PMC_R_TASK()
		42	Main Contactor Close			**Engine Controller Module**	
Vehicle Controller Sporadic Signals	19	31	Reverse and 2nd Gear Clutches		VCS_W_TASK()	**Transmission Control Unit**	VCS_R_TASK()
		34	DC/DC Converter Current Control			**BODY Control Module**	
		35	12V Power Relay				
		37	Brake Solenoid			**Hydraulic Brake Control Unit**	
		38	Backup Alarm				
		39	Warning Lights			**Engine Controller Module**	
		40	Key Switch				
		44	FWD/REV				
		46	Idle				

DOI:10.5963/IJCSAI0301002

		48	Shift in Progress				
		53	Main Contactor Acknowledge				
Contactor Control Signals	20	29	High Contactor Control		CCS_W_TASK()	**BODY Control Module**	CCS_R_TASK()
		30	Low Contactor Control				
Motor Controller Sporadic Signals	21	41	Main Contactor Close		MCS_W_TASK()	**Engine Controller Module**	MCS_R_TASK()
		45	FW/Rev Ack.				
		47	Inhibit				
		50	Inverter Temperature Status				
		51	Shutdown				
		52	Status/Malfunction				
Converter and Battery Test	35	33	DC/DC Converter		CBT_W_TASK()	**BODY Control Module**	CBT_R_TASK()
		36	Traction Battery Ground Fault Test			**Hydraulic Brake Control Unit**	
Brake Pressure Signals	2	8	Brake Pressure, Master Cylinder	**Hydraulic Brake Control Unit**	BPS_ W_TASK()	**Engine Controller Module**	BPS_ R_TASK()
		9	Brake Pressure, Line				
Brake Switch Signal	30	18	Brake Switch		BSS_ W_TASK()		BSS_ R_TASK()
Vehicle Speed Signal	32	12	Vehicle Speed		VSS_ W_TASK()	**Engine Controller Module**	VSS_ R_TASK()
						Traction Control Unit	
						ESP/ROM	
Transaction Pressure Signal	5	11	Transaction Clutch Line Pressure	**Transmission Control Unit**	TPS_ W_TASK()	**Engine Controller Module**	TPS_ R_TASK()
Transaxle Lubrication Pressure	33	10	Transaxle Lubrication Pressure		TLP_ W_TASK()		TLP_ R_TASK()
Motor/Trans Over Temperature	36	21	Motor/Trans Over Temperature		MOT_ W_TASK()		MOT_ R_TASK()
Front-Left wheel speed	9	67	Front-Left wheel speed	**Front-Left Wheel Module**	S_W_TASK()	**Active Frame Steering**	S_R_TASK()
						Electronic Brake Control Module	
Front-Left Wheel Suspension Sensing signals	23	54	Front-Left Suspension Deflection		SSS_W_TASK()	**Active Suspension Unit**	SSS_W_TASK()
		58	Front-Left Unsprung mass velocity				
Front-Right wheel speed	10	68	Front-Right wheel speed	**Front-Right Wheel Module**	S_W_TASK()	**Active Frame Steering**	S_R_TASK()
						Electronic Brake Control Module	
Front- Right Wheel Suspension Sensing signals	24	55	Front-Right Suspension Deflection		SSS_W_TASK()	**Active Suspension Unit**	SSS_R_TASK()
		59	Front-Right Unsprung mass velocity				
Rear-Left wheel speed	11	69	Rear-Left wheel speed	**Rear-Left Wheel Module**	S_W_TASK()	**Active Frame Steering**	S_R_TASK()
						Electronic Brake Control Module	
Rear-Left Wheel Suspension Sensing signals	25	56	Rear-Left Suspension Deflection		W_TASK()	**Active Suspension Unit**	SSS_R_TASK()
		60	Rear-Left Unsprung mass velocity				
Rear-Right wheel speed	12	70	Rear-Right wheel speed	**Rear-Right Wheel Module**	S_W_TASK()	**Active Frame Steering**	S_R_TASK()
						Electronic Brake Control Module	
Rear-Right Wheel Suspension Sensing signals	26	57	Rear-Right Suspension Deflection		SSS_W_TASK()	**Active Suspension Unit**	SSS_R_TASK()
		61	Rear-Right Unsprungmass velocity				

DOI:10.5963/IJCSAI0301002

Active Suspension Unit Control	27	62	Front-Left Control Force	**Active Suspension Unit**	C_W_TASK()	**Front-Left Wheel Module**	C_R_TASK()
		63	Front-Right Control Force			**Front-Right Wheel Module**	
		64	Rear-Left Control Force			**Rear-Left Wheel Module**	
		65	Rear-Right Control Force			**Rear-Right Wheel Module**	
Steering torque	22	66	Steering torque sensor	**Active Frame Steering**	STS_W_TASK()	**Front-Left Wheel Module**	STS_R_TASK()
						Front-Right Wheel Module	
						Rear-Left Wheel Module	
						Rear-Right Wheel Module	
Brake Control	14	72	Brake Control Command	**Electronic Brake Control Module**	BCS_W_TASK()	**Front-Left Wheel Module**	BCS_R_TASK()
						Front-Right Wheel Module	
						Rear-Left Wheel Module	
						Rear-Right Wheel Module	
Throttle opening	7	73	Throttle opening(angle)	**Throttle Control Unit**	TOS_W_TASK()	**Traction Control Unit**	TOS_R_TASK()
Throttle command	8	74	Throttle command	**Traction Control Unit**	TCS_W_TASK()	**Throttle Control Unit**	TCS_R_TASK()
Brakes command	15	75	Brakes command		BCS_W_TASK()	**Hydraulic Brake Control Unit**	BCS_R_TASK()
Brake pressure	16	81	Brake pressure command	**ESP/ROM**	BPS_W_TASK()	**Electronic Brake Control Module**	BPS_R_TASK()
ESP Signals	28	76	Vehicle lateral acceleration		ESP_W_TASK()	**Adaptive Cruise Control**	VAA_R_TASK()
		77	Vehicle longitudinal acceleration				
		78	Sideslip angle				
		79	Yaw rate				
		80	Roll Angle				
Adaptive Cruise Control	29	82	Engine throttle control command	**Adaptive Cruise Control**	ACC_W_TASK()	**Engine Controller Module**	ACC_R_TASK()
		83	Brakes control command			**Electronic Brake Control Module**	
		84	RADAR Distance			**Local Sensor**	

V. SCHEDULING PARAMETERS IN FLEXRAY NETWORKS

A. FlexRay Bus

FlexRay is a real-time communication bus [1] designed to operate at speeds of up to 10 M bits/s. He is being developed by a consortium that includes automobile builder. It offers time-triggered and an event triggered architecture. Data are transmitted in payload segment containing between 0 and 254 bytes of data, 5 bytes for the Header segment and 3 bytes for the trailer segment. The topology may be linear bus, star or hybrid topologies. This bus contains two channels; each node could be connected to either one or both channels.

FlexRay bus contains a static segment for time triggered messages and a dynamic segment for event triggered messages. In time triggered networks, nodes only obtain network access at specific time periods, also called time slots. In event triggered networks, nodes may obtain network access at any time instant. The static (ST) segment and the dynamic (DYN) segment lengths can differ, but are fixed over the cycles. Both the ST and DYN segments are composed of several slots. The first two bytes of the payload segment are called message ID, this is used only in dynamic segment. The message ID can be used as a filterable data.

In this paper we will study the transmission parameters of DDS nodes on a FlexRay bus. During any slot, only one node is allowed to send on the bus, and that is the node which holds the message with the frame identifier (Frame ID) equal to the current value of the slot counter. There are two slot counters, corresponding to the ST and DYN segments, respectively. The assignment of frame identifiers to nodes is static and decided offline, during the design phase. Each node that sends messages has one or more ST and /or DYN slots associated to it. The bus conflicts are solved by allocating offline one slot to at most one node, thus making possible for two nodes to send during the same ST and DYN slot. FlexRay allows the sharing of the bus among event driven (ET) and time driven (TT) messages.

DOI:10.5963/IJCSAI0301002

For a distributed system based on FlexRay, task scheduling can be SCS (Static Cyclic Scheduling) or FPS (Fixed Priority Scheduling). For the SCS tasks and ST messages, the schedule table could be built. For FPS tasks and DYN messages, the worst-case response times had to be determined.

B. Communication Cycle

The FlexRay protocol organizes time into communication cycles, every cycle is organized into four parts, segments of configurable duration: The static segment is used to send critical, real-time data, and is divided into static slots, in which the electronic control units (ECUs) can send a frame on the bus. These frames consist of a header, payload and trailer and are assigned to the slots according to a static, TDMA-based schedule. Channel idle time is enforced between frames to prevent overlapping consecutive frames. The dynamic segment enables event-triggered communication. The lengths of the mini slots in the dynamic segment depend on whether or not an ECU sends data. The symbol window is used to transmit special symbols, for example to start up the FlexRay cluster. The network idle time interval is used by the nodes to allow them to correct their local time bases in order to stay synchronized to each other.

The length of an ST slot is specified by the FlexRay global configuration parameter gdStaticSlot. The length of the DYN segment is specified in number of mini-slots gNumberOfMinislots.

C. Static Segment Parameters

In a general communication process, response time can be divided in four pieces, as shown in Fig.1; generation delay, queuing delay, transmission delay and reception delay [18].

Generation delay is started when the transmitting node received the request of sending from a frame until the data are written into the buffer and ready for being sent. Queuing delay is started when generation delay ended until the frame acquires the occupation of the bus and begins to be sent. Transmission delay is the time during which the frame is being transmitted on the bus. Reception delay is started when the frame gets off the bus and goes into the receiving node until the frame accomplishes its task.

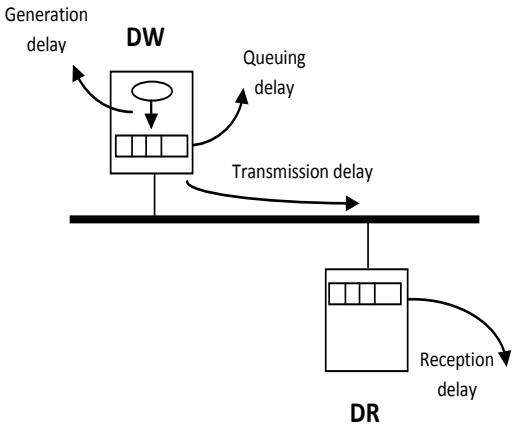

Fig. 1 Communication Model between DataReader and DataWriter

Note that the generation delay and reception delay are not related to the FlexRay network characteristics, but related to the given MCU performance. Therefore, these two parts of delay should not be taken into account. In FlexRay protocol the average response time R_m of a given frame is the sum of queuing delay average (t_m) and transmission delay average (C_m):

$$R_m = t_m + C_m \tag{1}$$

Since the static segment is transmitting at fixed time points in each FlexRay communication cycle without any queuing delays, the response time can be approximated by Cm.

$$R_m = C_m \tag{2}$$

Transmission delay Cm refers to the time interval between being on the bus and completion of sending process. It depends on the frame itself as well as bus parameters.

$$C_{m,s} = [TSS + FSS + FES + t_d + (HS + TS + S_m) \times (8 + BSS)]\tau_{bit} \tag{3}$$

TSS is the Transmission Start Sequence (3~15 bits). FSS is the Frame Start Sequence (1 bit). FES is the Frame End Sequence (2 bits). t_d is the delay related to sending and receiving nodes, which is around 2~3 bits. S_m represents the data field length (number of bytes) of the data frames. In addition, two BSS (Bit Start Sequence) are added before each byte. The constant "8" added to the data field length S_m refers to the sum of the FlexRay Header Segment (HS: 5) and Trailer Segment (TS: 3) lengths (number of bytes). Finally, τ_{bit} refers to the one bit transmission delay.

DOI:10.5963/IJCSAI0301002

D. Dynamic Segment Parameters

In the dynamic segment, the Time Division Multiple Access (TDMA) scheme is used for message scheduling. The Worst Case Response Time Rm of a queued data message m is defined as the longest time taken by the message to reach the destination station [19]. Then, for a dynamic (DYN) message m [20], the Worst Case Response Time is given by the following equation [21]:

$$\mathrm{R_m}(t) = \sigma_m + W_m(t) + C_m \tag{4}$$

Where Cm is the message communication time and σ_m is the longest delay endured during one bus cycle if the message is generated by its sender task after its slot has passed. Thus, in the worst case, the delay σ_m has the value:

$$\sigma_m = \mathrm{T_{bus}} - (\mathrm{ST_{bus}} + (\mathrm{FrameID_m} - 1) \times \mathrm{gdMinislot}) \tag{5}$$

Where T_{bus} is the length of the bus cycle, ST_{bus} is the ST segment length and gdMinislotis the smaller length of a slot when no message is to be sent during a particular slot.

Finally, Wm(t) is the worst case queuing delay caused by the transmission of ST frames and higher priority DYN messages during a given time interval t. Wm represents then the maximum amount of delay on the bus that can be produced by interference from ST Frames and DYN messages m. During the DYN slot, Frame ID_m can be delayed because of the following causes:

Local messages with higher priority, that uses the same frame identifier as m denoted by hp(m);

Any message in the system that can use DYN slots with lower frame identifiers than the one used by m, denoted by lf(m);

Unused DYN slots with frame identifiers lower than the one used for sending m. Such mini-slots are denoted by ms(m).

$$\mathrm{W_m}(t) = BusCycles_m(t) \times T_{bus} + W'_m(t) \tag{6}$$

Where BusCyclesm(t) is the number of bus periods for which the transmission of m is not possible because of messages transmission from hp(m), lf(m) and ms(m).

$$\begin{aligned} BusCycles_m(t) &= BusCycles_m(hp(m,t)) + \\ &BusCycles_m(lf(m,t), ms(m,t)) \end{aligned} \tag{7}$$

W'm(t) denotes the time that passes, in the last bus cycle, until m is sent which is measured from the beginning of the bus cycle, in which message m is sent, until the actual transmission of m starts.

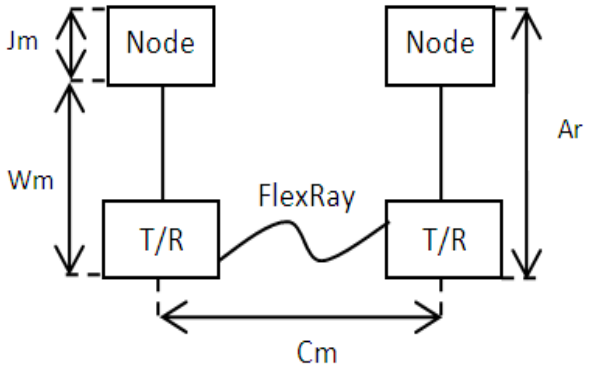

Fig. 2 Time decomposition of response time over FlexRay

Compared to static segment, dynamic segment has at least more 4 bits consisting in the DTS (Dynamic Trailing Sequence) that arranges the time slots at the end of each transmitted frame on the dynamic segment.

$$\begin{aligned} \mathrm{C_{m,s}} &= [TSS + FSS + FES + t_d + (HS + TS + S_m) \times \\ &(8 + BSS) + DTS]\tau_{bit} \end{aligned} \tag{8}$$

Where DTS is used to avoid an earlier detection of the channel idle by the receiver CAN. The DTS consists of two parts; a variable-length period at low level, followed by a fixed-length period at high level [1].

In our study, we will not ignore the scheduling process on the node, but we will consider its impact on the response time and evaluate its performance.

DOI:10.5963/IJCSAI0301002

VI. RESPONSE TIME CALCULATION

The full model is inspired from the FPS (First Priority Scheduling) approach, which is the most widely used approach in the computing world. In this case, each task has a fixed, static, priority, which is ECU pre-run-time. The runnable tasks are executed in the order determined by their priority, knowing that in real-time systems, the "priority" of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity.

The full model was conceived to be used in an industrial context[22], the temporal overheads of implementing the system must be taken into account such as:

- Context switches (one per job);
- Interrupts (one per sporadic task release);
- Real-time clock overheads.

In this case the Response time equation is rather than:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{9}$$

Where hp(i) is the set of tasks with priority higher than task i,C_i is the worst case computation time of the task iand $T_j$is the minimum time between task releases, jobs or task period.

The new equation is:

$$R_i = CS^1 + C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (CS^1 + CS^2 + C_j) \tag{10}$$

Where the new terms$CS^1$and $CS^2$are the cost of switching tothe task, and the cost of switching away from it. And the term $B_i$is the cost of the task worst case blocking time.

The cost of handling interrupts is as flowing:

$$\sum_{k \in \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH \tag{11}$$

Where Γ_s is the set of sporadic tasks and, IH is the cost of a single interrupt (which occur sat maximum priority level).

There is also a cost per clock interrupt, a cost for moving one task from delay to run queue and a (reduced) cost of moving groups of tasks

Let CT_cbe the cost of a single clock interrupt, $\Gamma_p$be the set of periodic tasks, and CT_s be the cost of moving onetask the following equation can be derived

$$R_i = CS^1 + C_i + B_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \left(CS^1 + CS^2 + C_j\right) + \sum_{k \epsilon \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH + \left\lceil \frac{R_i}{T_{clk}} \right\rceil CT_c + \sum_{g \epsilon \Gamma_p} \left\lceil \frac{R_i}{T_g} \right\rceil CT_s \tag{12}$$

A. Full Model Applied on the Static Segment Tasks

Within the static segment a static time division multiple access scheme is applied to coordinate transmissions. In the static segment all communication slots are of identical, statically configured duration and all frames are of identical, statically configured length. In order to schedule transmissions each node maintains a slot counter state variable vSlotCounter for Channel A and a slot counter state variable vSlotCounter for Channel B. Both slot counters are initialized with 1 at the start of each communication cycle and incremented at the end boundary of each slot.

In the Implementations of the FlexRay bus, the periodic and safety-criticaldata arescheduled on the static time-triggered segment, so the tasks in the static segment are periodic tasks that have the same priority per communication cycle.

Considering these facts, the Equation (12) applied on the static segment context becomes:

$$R_i = CS^1 + C_i + B_i + + \sum_{k \epsilon \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH + \left\lceil \frac{R_i}{T_{clk}} \right\rceil CT_c + \sum_{g \epsilon \Gamma_p} \left\lceil \frac{R_i}{T_g} \right\rceil CT_s \tag{13}$$

B. Full Model Applied on the Dynamic Segment Tasks

Within the dynamic segment a dynamic mini-slotting based scheme is used to arbitrate transmissions. In the dynamic segment the duration of communication slots may vary in order to accommodate frames of varying length. In order to schedule transmissions each node continues to maintain the two slot counters - one for each channel - throughout the dynamic segment. While the slot counters for Channel A and for Channel B are incremented simultaneously within the static segment, they may be incremented independently according to the dynamic arbitration scheme within the dynamic segment.

DOI:10.5963/IJCSAI0301002

In the Implementations of the FlexRay bus, the dynamic segment is mainly used for maintenance anddiagnosis data so the tasks are event triggered sporadic tasks that have different priority by bus communication cycle.

Considering these facts the Equation (12) applied on the static segment context becomes:

$$R_i = CS^1 + C_i + B_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \left(CS^1 + CS^2 + C_j\right) + \sum_{k \epsilon \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH + \left\lceil \frac{R_i}{T_{clk}} \right\rceil CT_c + \sum_{g \epsilon \Gamma_p} \left\lceil \frac{R_i}{T_g} \right\rceil CT_s \quad (14)$$

VII. The Studied Architecture

To illustrate the utility of our Comprehensive Scheduling Strategy(CSS),we have chosen to work within a platform of a vehicular network based on the SAE standard. In this system, a set of network processors subsystems produces routing data. These data mustbe distributed along the vehicular network. In previous works[14], we have attempted to implement this system using a combined scheduling but this kind of implementation introduced complexity and is inappropriate in case of automotive systems.

The main goal of the proposed architecture is to reduce this complexity when adopting the data-centric publish- subscribe paradigm which is very suitable for real-time communication along with a FPS full model scheduling used to schedule the communicating tasks on the node. The framework architecture is a set of nodes formed by Fujitsu cards with an MB91F465X controller, connected via a Real-Time Transport protocol. Each node is embedded a Real-Time Operating System µ COSII, a middleware,andaPublish-SubscribeinterfaceaccordingtoDDSspecification, as represented in Fig.3.

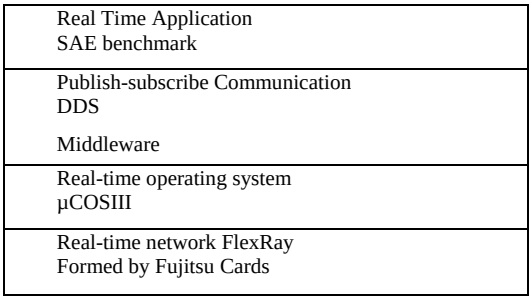

Fig. 3 Architecture of distributed real-time system using a publish-subscribe paradigm

The communication between nodes is achieved due to publish subscribe interface via the Global Data Space that is represented by arelational data model. The middle ware has to keep track of the data objects instances, which are considered as rows in a table.Eachdataobjectisidentifiedbythecombinationofatopicandatopicspecifiedkey.

The results obtained in this paper are a based on a simulation under Matlab, using these environment parameters.

VIII. QOS APPROXIMATION

In this paper we will use the calculated response time to evaluate if the DDS QoS real-time parameters can be met in the SAE benchmark application context. We will focus our interest on two real-time policies, the Deadline QoS policy represented by the parameter deadline period of a task, and Time Based Filter Policy represented by the parameter minimum_seperation period.

A. Deadline QoS Policy

This policy is used for cases where a Topic(i.e., SAE benchmark application Topic), is expected to have each instance updated periodically. On the publishing side this setting establishes a contract that the application must meet. On the subscribing side the setting establishes a minimum requirement for the remote publishers that are expected to supply the data values [2]. When the Service 'matches' a DataWriter and a DataReader it checks whether the settings are compatible (i.e., offered deadline period<= requested deadline period). Assuming that the reader and writer ends have compatible settings, the fulfillment of this contract is monitored by the Service and the application is informed of any violations by means of the proper listener or condition. The value offered is considered compatible with the value requested if and only if the inequality "offered deadline period <= requested deadline period" evaluates to 'TRUE.' The setting of the DEADLINE policy must be set consistently with that of the TIME BASED FILTER. For these two policies to be consistent the settings must be such that "deadline period>= minimum_separation", and the deadline period should always be inferior to the application period T.

$$T \geq D \geq Min\_Sep \quad (15)$$

On the other hand, the response time of the application should not exceed its deadline to guarantee the freshness of the data sample.

$$D \geq R \quad (16)$$

DOI:10.5963/IJCSAI0301002

B. Time Based Filter QoS Policy

This policy allows a DataReader to indicate that it does not necessarily want to see all values of each instance published under the Topic. Rather, it wants to see at most one change every minimum_separationperiod. The TIME BASEDFILTER is applied to each instance of the data separately, that is, the constraint is that the DataReader does not want to see more than one sample of each instance per minumum_separation period. This setting allows a DataReader to further decouple itself from the DataWriter objects. It accommodates the fact that for fast-changing data different subscribers may have different requirements as to how frequently they need to be notified of the most current values. The TIME BASED FILTER specifies the samples that are of interest to the DataReader [2]. The setting of the TIME BASED FILTER minimum_separation must be consistent with the deadline period. For these two QoS policies to be consistent they must verify that "period >= minimum_separation".

On the other hand, in order to verify the Time Based Filter QoS policy the response time of the application should be superior to the minimum_seperation period to guarantee that theDataWriter doesn't produce data faster than the DataReader consumes it.

$$R \geq Min\_Sep \tag{17}$$

To resume, we can validate these two QoS policies by following this equation:

$$T \geq D \geq R \geq Min\_Sep \tag{18}$$

IX. RESULTS AND COMMENTS

In this section, we propose an algorithm to calculate the response time of the DataWariters tasks and evaluate the DDS QoS based on the generated values.

The Equations (13) and (14) give us the needed parameters to determine the response time for both static and dynamic segments tasks:

- C_i, the computing time is equivalent to the transmission delay $C_{m,s}$ and$C_{m,d}$, because the execution of a message relative to a writing task is the fact to transmit data on the bus.
- The worst blocking time Bi is defined as follows:

$$B_i = \frac{\text{Frame size} - 1 \text{ bit}}{\text{Lowest flow rate used}} \tag{19}$$

This equation is true for the CAN case but in the FlexRay case:

$$B_i = 0$$

- $CS^1$is the cost of switching to the task, this parameter is given by the used real-time operating system µCOSIII [23].

$$CS^1 = 0.005 \text{ ms}$$

- $CS^2$is the cost of switching away from the task, this parameter is also given by the used real-time operating system µCOSIII[23].

$$CS^2 = 0.009 \text{ ms}$$

- IHis the cost of executing an interrupt service routine which occurs at maximum priority level, the more there are STATUS register in the system, the handler time to execute the interrupt routine is longer, the FlexRay driver interrupt routine is the longest on the status of receipt of communications data. For our study we approximate this parameter as follow:

$$IH = 10 \times CT_c$$

- $CT_c$is the cost of a single clock interrupt for the microcontroller MB91F465X we have approximated its value:

$$CT_c = \frac{1}{10} \times T_{clk}$$

- $CT_s$is the cost of moving one task, which is equivalent to switching a task.

$$CT_s = CS^2$$

- $T_{clk}$is the clock period calculated for a given core frequency.

The response time calculation process is described by the following algorithm:

Algorithm Worst Case Response Time Computing

DOI:10.5963/IJCSAI0301002

```
for iin1..N loop
  n := 0
  loop
    Calculate Ci for periodic tasks
    Calculate Ci for sporadic tasks
    n := n + 1
  end loop
end loop
for iin1..N loop
  n := 0
W_i^n = C_i
  loop
    calculate new w_i^{n+1}
    if w_i^{n+1} = w_i^n then R_i = w_i^n
exit value found
end if
if w_i^{n+1} > T_i then
exit value notfound
end if
    n := n + 1
end loop
end loop
```

For the simulation, we consider a set of FlexRay nodes the sending 36 messages on the FlexRay bus. Since each node in the system that generates static messages that needs at least one static slot, the minimum number of static slots is the number of nodes (nodesST) that send static messages [1]. In the extended benchmark [17], there are 15 nodes sending 36 messages among which there are 30 periodic messages that need to be scheduled on the FlexRay static segment, but we will regroup them into 6 nodes. The remaining six messages are sporadic and needs to be mapped into the dynamic segment.

The period of the bus cycle (gdCycle) must be lower than the maximum cycle length cdCycleMax equal to 16 ms and has, also, to be an integer divisor of the period of the global static segment. In addition, each node has a counter vCycleCounter in the interval [0…63]. Thus, during a period of the global static schedule there can be at most 64 bus cycles, observing our message set, we have noticed that almost all of the message periods are multipliers of 5ms; So we can fix the period of the bus cycle to 5 ms and adjust some message periods, especially the messages introduced by Ben Gaid, M-M in [24] and others introduced by M. Utayba in [17]. All messages with period equal to 8 ms will have a new period of 5 ms, and messages with period equal to 12 ms will have 10 ms as new period, as the bus cycle length is equal to 5 ms the distance between each data sample, or, period should be 5ms or a multiple of it. This will not affect our system efficiency since it will make it faster and more reactive. There is another problem with messages with a 1000 msperiod that are not schedulable. In fact, even if we consider the longest period of the global static schedule (64 bus cycles), we wouldn't manage to reach the 1000 ms. Thus, we have to decrease this period to 64*5=320 ms. We have also replaced the original bus priorities designed for an event triggered bus (CAN) by a local priority able to order transmission of messages having the same Frame Identifier on different slots assigned to their source node.

Applying the previous algorithm with 3 different bus speeds; 5 M bit/s and 10 M bit/s for one channel transmission scheme, and 20 M bit/s for both channels transmission scheme, we obtain the following results.

TABLE II.BODY CONTROL MODULE RESULTS

Vehicle Module	Message ID	Size (Bytes)	Deadline [ms]	Min Separation [ms]			T [ms]	Task Priority	Worst Case Response Time R (ms)					
									Bus Speed 5Mbit/s		Bus Speed 10 Mbit/s		Bus Speed 20 Mbit/s	
				S1	S2	S3			Core F 12 Mhz	Core F 100 Mhz	Core F 12 Mhz	Core F 100 Mhz	Core F 12 Mhz	Core F 100 Mhz
BODY Control Module	1	1	5	0.0214	0.0107	0.0054	50	2	0.4502	0.1629	0.2795	0.1010	0.1941	0.0701
	3	1	5	0.0206	0.0103	0.0052	5	1	0.1706	0.0616	0.1397	0.0504	0.1242	0.0448
	13	1	**5**	0.0206	0.0103	0.0052	**5**	1	0.1706	0.0616	0.1397	0.0504	0.1242	0.0448
	17	1	10	0.0214	0.0107	0.0054	10	3	0.5144	0.1861	0.3115	0.1126	0.2102	0.0759
	18	2	10	0.0234	0.0117	0.0059	10	4	0.5846	0.2116	0.3467	0.1254	0.2277	0.0823
	31	4	100	0.0266	0.0133	0.0067	100	1	0.1886	0.0681	0.1487	0.0536	0.1287	0.0464
	34	3	**320**	0.0246	0.0123	0.0062	**320**	1	0.1825	0.0659	0.1457	0.0525	0.1272	0.0458

TABLE III.ENGINE CONTROLLER MODULE RESULTS

Vehicle Module	Message ID	Size (Bytes)	Deadline [ms]	Min Separation [ms]			T [ms]	Task Priority	WORST CASE RESPONSE TIME R (MS)					
									Bus Speed 5 Mbit/s		Bus Speed 10 Mbit/s		Bus Speed 20 Mbit/s	
				S1	S2	S3			Core F 12 Mhz	Core F 100 Mhz	Core F 12 Mhz	Core F 100 Mhz	Core F 12 Mhz	Core F 100 Mhz
Engine Controller	4	2	5	0.0226	0.0113	0.0057	5	1	0.1763	0.0637	0.1424	0.0514	0.1255	0.0453

DOI:10.5963/IJCSAI0301002

Module	6	2	5	0.0226	0.0113	0.0057	5	1	0.1763	0.0637	0.1424	0.0514	0.1255	0.0453
	19	6	10	0.0314	0.0157	0.0079	10	2	0.4679	0.1694	0.2882	0.1043	0.1984	0.0717
	20	2	10	0.0226	0.0113	0.0057	10	1	0.1763	0.0637	0.1424	0.0514	0.1255	0.0453
	21	3	10	0.0254	0.0127	0.0064	10	3	0.5441	0.1970	0.3263	0.1181	0.2174	0.0786
	35	1	**320**	0.0206	0.0103	0.0052	**320**	1	0.1703	0.0615	0.1394	0.0503	0.1240	0.0447

TABLE IV.CENTRAL CONTROL UNIT RESULTS

Vehicle Module	**Message ID**	**Size (Bytes)**	**Deadline [ms]**	**Min Separation [ms]**			**T ms]**	**Task Priority**	**Worst Case Response Time R (ms)**					
									Bus Speed 5Mbit/s		**Bus Speed 10 Mbit/s**		**Bus Speed 20 Mbit/s**	
				S1	**S2**	**S3**			**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**
Active Suspension Unit	27	2	**10**	0.0226	0.0113	0.0057	**10**	1	0.2568	0.0930	0.2229	0.0808	0.2060	0.0746
Active Frame Steering	22	2	10	0.0226	0.0113	0.0057	10	1	0.2568	0.0930	0.2229	0.0808	0.2060	0.0746
Electronic Brake Control Module	14	4	**5**	0.266	0.0133	0.0067	**5**	1	0.2688	0.0974	0.2289	0.0829	0.2090	0.0757
Traction Control Unit	8	1	5	0.0206	0.0103	0.0052	5	1	0.2508	0.0909	0.2199	0.0797	0.2045	0.0741
	15	4	**5**	0.266	0.0133	0.0067	**5**	1	0.2688	0.0974	0.2289	0.0829	0.2090	0.0757
ESP/ROM	16	4	**5**	0.266	0.0133	0.0067	**5**	1	0.2688	0.0974	0.2289	0.0829	0.2090	0.0757
	28	5	**10**	0.286	0.143	0.0072	**10**	1	0.2748	0.0996	0.2319	0.0840	0.2105	0.0762

TABLE V.FRONT CONTROL UNIT

Vehicle Module	**Message ID**	**Size (Bytes)**	**Deadline [ms]**	**Min Separation [ms]**			**T [ms]**	**Task Priority**	**Worst Case Response Time R (ms)**					
									Bus Speed 5Mbit/s		**Bus Speed 10Mbit/s**		**Bus Speed 20Mbit/s**	
				S1	**S2**	**S3**			**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**
Hydraulic Brake Control Unit	2	2	5	0.0226	0.0113	0.0057	5	1	0.2838	0.1028	0.2499	0.0905	0.2329	0.0844
	30	1	20	0.0206	0.0103	0.0052	50	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
	32	1	100	0.0206	0.0103	0.0052	100	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
Transmission Control Unit	5	1	5	0.0206	0.0103	0.0052	5	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
	33	1	100	0.0206	0.0103	0.0052	100	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
	36	1	**320**	0.0206	0.0103	0.0052	**320**	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
Throttle Control Unit	7	1	5	0.0206	0.0103	0.0052	5	1	0.2778	0.1007	0.2469	0.0895	0.2315	0.0839
Adaptive Cruise Control	29	3	10	0.0246	0.0123	0.0062	10	1	0.2898	0.1050	0.2529	0.0916	0.2344	0.0849

TABLE VI.RIGH WHEEL UNIT

Vehicle Module	**Message ID**	**Size (Bytes)**	**Deadline [ms]**	**Min Separation [ms]**			**T [ms]**	**Task Priority**	**Worst Case Response Time R (ms)**					
									Bus Speed 5Mbit/s		**Bus Speed 10 Mbit/s**		**Bus Speed 20 Mbit/s**	
				S1	**S2**	**S3**			**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**
Front-Right Wheel Module	10	1	**5**	0.0206	0.0103	0.0052	**5**	1	0.1698	0.0615	0.1389	0.0503	0.1235	0.0447
	24	2	**10**	0.0226	0.0113	0.0057	**10**	1	0.1758	0.0637	0.1419	0.0514	0.1250	0.0453
Rear-Right Wheel Module	12	1	**5**	0.0206	0.0103	0.0052	**5**	1	0.1698	0.0615	0.1389	0.0503	0.1235	0.044
	26	2	**10**	0.0226	0.0113	0.0057	**10**	1	0.1758	0.0637	0.1419	0.0514	0.1250	0.0453

TABLE VII.LEFT WHEEL UNIT

Vehicle Module	**Message ID**	**Size (Bytes)**	**Deadline [ms]**	**Min Separation [ms]**			**T [ms]**	**Task Priority**	**Worst Case Response Time R (ms)**					
									Bus Speed 5Mbit/s		**Bus Speed 10 Mbit/s**		**Bus Speed 20 Mbit/s**	
				S1	**S2**	**S3**			**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**	**Core F 12 Mhz**	**Core F 100 Mhz**
Front-Left Wheel Module	10	1	**5**	0.0206	0.0103	0.0052	**5**	1	0.1698	0.0615	0.1389	0.0503	0.1235	0.0447
	24	2	**10**	0.0226	0.0113	0.0057	**10**	1	0.1758	0.0637	0.1419	0.0514	0.1250	0.0453
Rear-Left Wheel Module	12	1	**5**	0.0206	0.0103	0.0052	**5**	1	0.1698	0.0615	0.1389	0.0503	0.1235	0.044
	26	2	**10**	0.0226	0.0113	0.0057	**10**	1	0.1758	0.0637	0.1419	0.0514	0.1250	0.0453

DOI:10.5963/IJCSAI0301002

We have regrouped in the Central Control Unit, Table V all the critical modules and signal responsible for the vehicle control.

In the Front Control Unit, Table VI, we have regrouped all the modules and signals responsible for the cruise control.

The Right Wheel Unit and the Left Wheel Unit contains the modules and signals of respectively right and left sides of the vehicle.

We have used for the simulation two different Core frequencies, the best case Core frequency which is 100 Mhz and the worst case Core frequency which is 12 Mhz. For both cases, we have noticed that the deadline has been met and the equation below is verified.

$$T \geq D \geq R$$

Thanks to FlexRay bus speed, we can assume that the DDS Deadline QoS Policy always be reached.

As for the Time Based Filter we have approximated the minimum_separation parameter to be the reception delay which is for FlexRay case the transmission delay Cm. So for each bus speed we have a different value for the minimum_separation period.

Same as the DDS Deadline QoS Policy, we can assume that the Time Based Filter QoS Policy is verified.

$$R \geq Min\_Sep$$

For a fixed bus speed and two different frequencies the response time is the lowest when the Core frequency is the highest. This is a logical result the ECU in this case can treat the tasks faster and consequently the response time has a lower value represented by the execution time in the full model equation.

For a fixed frequency and a variable bus speed the worst case response time is the lowest when the bus speed is the highest. This is a logical result as the response time is dependent of the bus speed.

X. CONCLUSION

The DDS middleware, based on the publish-subscribe paradigm, is very suitable for real-time distributed communication systems, allowing the distribution of hard real-time applications. In this paper, we have proposed to use DDS on top of the real-time network FlexRay to take advantage of its high speed and to profit of the DDS QoS management in an automotive context. We have proposed a scheduling model based full FPS scheduling to first calculate the worst case response time for our vehicular system and evaluate its performance on a benchmark application, an extended SAE benchmark. After the simulations, results have shown that not only the applications QoS requirement has been met but also this combination is very appropriate for the vehicular context. One promising research direction would be the evaluation of the rest of real-time QoS parameters offered by DDS on the same system configuration and the extension of DDS to be integrated with control engineering domain.

In fact, DDS is not yet mature for use in vehicle domain without some extension like DDS ports and connectors for DDS software composition, the mapping of DDS based components to Simulink variant blocks, the converting of Simulink models with DDS to an AADL framework and integration of behavioral implementations. The implementation of an authoring tool for DDS is also required. DDS has not specifications for the integration of vehicle networks like CAN, LIN and FlexRay. All these points must be clarified in the future. We plan to have a complete platform to argue our future propositions to improve the actual situation.

ACKNOWLEDGMENT

The researches presented in this paper would not have been possible without the support of many people. We wish to express our gratitude to the SYSCOM ENIT members for their help and assistance.

REFERENCES

[1] FlexRay Consortium, FlexRay Communications System-Protocol Specification, Version 2.1, Revision A, 2005.

[2] Object Management Group, Data distribution service for real-time systems, version 1.2, Massachusetts: Needham, January 2007.

[3] T. Guesmi, R. Rekik, S. Hasnaoui, H. Rezig, “Design and Performance of DDS-based Middleware for Real-Time Control Systems,”International Journal of Computer Science and Network SecurityIJCSNS, twelvethed, vol.7, 2007, pp 188-200.

[4] G. P. -Castellote, B.Farabaugh, R. Warren, “An Introduction to DDS and Data-Centric Communications,” Real-Time Innovations, Available http://www.rti.com.

[5] Prism Tech, “Opensplice C reference guide”, version 2.2, Massachusetts: Burlington, 2006, pp. 22-25.

[6] H. V. Hag, OpenSlice Overview, white paper, 2006.

[7] Thales Netherland, http://www.thales-nederland.nl/.

DOI:10.5963/IJCSAI0301002

[8] J. H. van 't Hag Data-Centric to the Max - The SPLICE Architecture Experience, Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), May 19 - 22, 2003.

[9] Christopher A. Lupini, "Vehicle Multiplex Communication - Serial Data Networking Applied to Vehicular Engineering", SAE, April 2004.

[10] Tindel, K., Burns, A., "Guaranteeing Message Latencies On Control Area Network (CAN)", Real-Time Systems Research Group, Department of Computer Science, University of York, England, 1994. [Online] available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.3545&rep =rep1&type=pdf.

[11] Kopetz, H., "A solution to an automotive Control System Benchmark", Real-Time Systems Symposium, 7-9 Dec. 1994, pp. 154-158.

[12] T. Guesmi, R. Rekik, S. Hasnaoui, H. Rezig, Design and Performance of DDS-based Middleware for Real-Time Control Systems, IJCSNC, VOL.7, No.12, 2007, pp 188-200.

[13] Rim Bouhouch, WafaNajjar, HoudaJaouani, Salem Hasnaoui, Implementation of Data Distribution Service Listeners on Top of FlexRay Driver, INFOCOMP 2011 : The First International Conference on Advanced Communications and Computation, IARIA, 64-69.

[14] W. Najjar, R. Bouhouch, H. Jaouani, S. Hasnaoui, "Static and Dynamic Scheduling for FlexRay Network Using the Combined Method", International Journal of Information Technology and Systems, Vol. 1, No. 1, pp. 18-26, January 2012.

[15] K. W. Tindell, A. Burns, A. J. WELLINGS; An extendible approach for analyzing fixed priority hard real-time tasks, Real_Time Systems, Vol.6, No.2, 1994, pp.133-151.

[16] A. Hagiescu, U. Bordoloi, S. Chakraborty, Performance Analysis of FlexRay-based ECU Networks; DAC proceedings, 2007, pp.284-289.

[17] M. Utayba Mohammad, N. Al-Holou, "Development of An Automotive Communication Benchmark", Canadian Journal on Electrical and Electronics Engineering, Vol. 1, No. 5, August 2010.

[18] T. Guangyn, B. Peng, C. Quanshi, Response Time Analysis of FlexRay Communication in Fuel Cell Hybrid Vehicle, Vehicle Power and Propulsion Conference VPPC'08. IEEE, 2008, pp. 1-4.

[19] P. Pop, P. Eles and Z. Peng, Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems, IEEE: Real-Time Computing Systems and Applications, RTCSA '99, Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306), 1999, pp. 287-294.

[20] N. Navet, Y. Song, F. Simonot-lion, C. Wilwert, Trends in automotive communication systems, Proceedings of the IEEE, Vol.93, No.6, 2005, pp. 1204-1223.

[21] T. Pop, P. Pop, P. Eles, Z. Peng, Timing Analysis of the FlexRay communication Protocol; Real-Time Syst, Vol.39, 2008, pp. 205-235.

[22] A. Burns, A. Wellings, "Scheduling Real-Time Systems", Chapter 11, Real-Time Systems and Programming Languages,The university of York, Department of Computer Science.

[23] A J. J. Labrosse. "MicroC/OS-II The Real Time Kernel". Miller Freeman, Inc, United States of America, 1999.

[24] Ben Gaid, M-M, Cela, A, Diallo, S., Kocik, R., Hamouche, R. and Reama, A, "Performance Evaluation of the Distributed Implementation of a Car Suspension System", In Proceedings of the IFAC Workshop on Programmable Devices and Embedded Systems, February 2006.

APPENDIX

We have added Table VII and Table IX taken from [17] to identify the extended SAE benchmark new signals and parameters used in our approach and simulation.

TABLE VIII.ASUMMARYOFALLSIGNALSINTHEEXTENDED BENCHMARK

Module	SignalNumber	SignalDescription	Size[bit]	Period[msec]	Deadline[msec]
Body Control Module(BCM)	1	Traction Battery Voltage	8	100	100
	2	Traction Battery Current	8	100	100
	3	TractionBatteryTemp, Average	8	1000	1000
	4	Auxiliary Battery Voltage	8	100	100
	5	TractionBatteryTemp, Max	8		1000
	6	Auxiliary Battery Current	8	100	100
	13	Traction Battery GroundFault	1	1000	1000
	14	Hi&Lo Contactor Open/Close	4	50/S	5
	23	12V Power Ack VehicleControl	1	50/S	20
	24	12V Power Ack Inverter	1	50/S	20
	25	12V Power AckI/M Control	1	50/S	20
	28	Interlock	1	50/S	20
	7	AcceleratorPosition	8	5	5

DOI:10.5963/IJCSAI0301002

	15	KeySwitchRun	1	50/S	20
	16	KeySwitchStart	1	50/S	20
	17	AcceleratorSwitch	2	50/S	20
	19	EmergencyBrake	1	50/S	20
	20	ShiftLever(PRNDL)	3	50/S	20
	22	Speed Control	3	50/S	20
	26	Brake Mode (Parallel/Split)	1	50/S	20
	27	SOCReset	1	50/S	20
	71	BrakePedal Position	8	8	8
EngineControlModule (ECM)	29	HighContactorControl	8	10	10
	30	LowContactorControl	8	10	10
	31	Reverse and 2nd Gear Clutches	2	50/S	20
	32	ClutchPressureControl	8	5	5
	33	DC/DCConverter	1	1000	1000
	34	DC/DC Converter CurrentControl	8	50/S	20
	35	12VPowerRelay	1	50/S	20
	36	Traction Battery Ground FaultTest	2	1000	1000
	37	BrakeSolenoid	1	50/S	20
	38	Backup Alarm	1	50/S	20
	39	WarningLights	7	50/S	20
	40	KeySwithc	1	50/S	20
	42	MainContactorClose	8	5	5
	44	FWD/REV	1	50/S	20
	46	Idle	1	50/S	20
	48	ShiftinProgress	1	50/S	20
	53	Main Contactor Acknowledge	1	50/S	20
	41	MainContactorClose	1	50/S	20
	43	TorqueMeasured	8	5	5
	45	FW/RevAck.	1	50/S	20
	47	Inhibit	1	50/S	20
	49	ProcessedMotorSpeed	8	5	5
	50	Inverter Temperature Status	2	50/S	20
	51	Shutdown	1	50/S	20
	52	Status/Malfunction	8	50/S	20
HydraulicBrakeControlUnit (HBCU)	8	BrakePressure, Master Cylinder	8	5	5
	9	BrakePressure,Lines	32	5	5
	12	VehicleSpeed	8	100	100
	18	BrakeSwitch	1	20/S	20
Transmission ControlModule(TCM)	10	TransaxleLubrication Pressure	8	100	100
	11	TransactionClutchLine Pressure	8	5	5
	21	Motor/Trans Over Temperature	2	1000	1000
Front-LeftWheel Module	54	Front-LeftSuspension Deflection	8	12	12
	58	Front-Left Unsprung MassVelocity	8	12	12
	67	Front-Leftwheelspeed	8	8	8
Front-RightWheel Module	55	Front-RightSuspension Deflection	8	12	12
	59	Front-RightUnsprung MassVelocity	8	12	12
	68	Front-Right wheel Speed	8	100	100
Rear-LeftWheel Module	56	Rear-Left Suspension Deflection	8	12	12
	60	Rear-Left Unsprung MassVelocity	8	12	12
	69	Rear-Leftwheelspeed	8	8	8
Rear-RightWheel Module	57	Rear-RightSuspension Deflection	8	12	12
	61	Rear-Right Unsprung MassVelocity	8	12	12
	70	Rear-Rightwheelspeed	8	8	8
ActiveSuspension	62	Front-Left Control	8	12	12

DOI:10.5963/IJCSAI0301002

Unit		Force			
	63	Front-Right Control Force	8	12	12
	64	Rear-LeftControl Force	8	12	12
	65	Rear-Right Control Force	8	12	12
ActiveFrame Steering(AFS)	66	Steeringtorquesensor	16	10	10
ElectronicBrake ControlModule (EBCM)	72	Brake Control Command	32	8	8
ThrottleControl Unit	73	Throttleopening	8	5	5
TractionControl Unit	74	Throttlecommand	8	5	5
	75	Brakes command	32	8	8
ESP/ROM	76	Vehicle lateral acceleration	8	12	12.5
	77	Vehicle longitudinal acceleration	8	12	12.5
	78	Sideslipangle	8	12	12.5
	79	Yawrate	8	12	12.5
	80	RollAngle	8	12	12.5
	81	Brake pressure command	32	8	8
AdaptiveCruise Control	82	Enginethrottlecontrol command	8	12	12.5
	83	Brakes control command	8	12	12.5
	84	RADARDistance	8	12.5	12.5

Rim Bouhouchreceived the Engineering degree in Telecommunications Engineering from the National Engineering School of Tunis, Tunisia, in 2008, a master degree in Communication Systems in the same institute, in 2009, and is a PhD student in Telecommunications. Her current research interests are the implement and evaluation of Real-time Middleware for Real-time vehicular networks.

HoudaJaouani received the engineering degree in Electrical Engineering from the National Engineering School of Tunis, Tunisia, in 2007, a master degree in Automatic and Signal Processing in the same institute, in 2008, and is a PhD student in Telecommunications. Her current research interests are performance evaluation of the DDS Middleware on Real-time vehicular networks using the SAE Benchmark as a standard vehicle prototype.

Amel Ben Ncira received the engineering degree in Electrical Engineering from the National Engineering School of Tunis, Tunisia, in 2005, a master degree in electrical systems in the same institute, in 2006, and is a PhD student in Electrical Engineering.

Salem Hasnaoui is a professor in the Department of Computer and Communication Technologies at the National School of Engineering of Tunis. He received the Engineer diploma degree in electrical and computer engineering from National School of Engineering of Tunis. He obtained a M.Sc. and third cycle doctorate in electrical engineering, in 1988 and 1993 respectively. The later is extended to a PhD. degree in telecommunications with a specialization in networks and real-time systems, in 2000. He is author and co-author of more than 200 refereed publications, a patent and a book. His current research interests include real-time systems, sensor networks, QoS control & networking, adaptive distributed real-time middleware and protocols that provide performance assured services in unpredictable environments. Prof. Hasnaoui is the responsible of the research group "Networking and Distributed computing" within the Communications Systems Laboratory at the National School of Engineering of Tunis. He served on many conference committees and journals reviewing processes and he is the designated inventor of the Patent "CAN Inter-Orb protocol-CIOP and a Transport Protocol for Data Distribution Service to be used over CAN, TTP and FlexRay protocols".

DOI:10.5963/IJCSAI0301002