ESPRIT: Overview of the Vehicles Road-Train Real-Time Architecture

Nicolas Gobillot, Eric Lucet

To cite this version:

Nicolas Gobillot, Eric Lucet. ESPRIT: Overview of the Vehicles Road-Train Real-Time Architecture. ERTS 2018, Jan 2018, Toulouse, France. hal-02156364

HAL Id: hal-02156364
https://hal.science/hal-02156364

Submitted on 14 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ESPRIT: Overview of the Vehicles Road-Train Real-Time Architecture

Nicolas Gobillot
CEA, LIST, Interactive Robotics Laboratory
91191 Gif-sur-Yvette, France
Email: nicolas.gobillot@cea.fr

Eric Lucet
CEA, LIST, Interactive Robotics Laboratory
91191 Gif-sur-Yvette, France
Email: eric.lucet@cea.fr

***Abstract*—This paper deals with the communication and real-time architecture of an innovative articulated vehicle, coupled with up to seven other similar vehicles to form a road-train. The controller device was chosen powerful and flexible enough to prototype an innovative system of vehicles able to reconfigure themselves after a coupling or uncoupling process. Automotive industry standards are consider in order to benefit from existing systems. Implementation on real vehicles demonstrates performances and robustness in scenario conditions.**

Keywords: Distributed architecture, real-time system, embedded system, vehicle automation.

Category: Abstract of regular paper

I. Introduction

The Easily diStributed Personal RapId Transit [3], [5] project was launched in order to design a new specific light weight electric vehicle able to be stacked together to gain space for improving car-sharing public transport. By using innovative coupling systems, up to eight ESPRIT vehicles can be nested together in a road-train, (see figure 1) seven being towed, for an efficient redistribution by a single operator. Then, ESPRIT vehicles are available to users at specifically designed charging stations scattered across cities.

Fig. 1. Two of the ESPRIT vehicle mock-ups.

A. Context

The ESPRIT vehicle design has been established in view of a future industrialization with the automotive industry. On the hardware point of view, most of the selected components are well tested and validated off-the-shelf modules that are already used in current cars and road vehicles. This design choice implies that the communication architecture between the different modules and between the different vehicles in road-train configuration is based on the Controller Area Network (CAN) bus, similarly to the one presented in [4]. On the software side, most of the calculators are industrial-grade controllers using proven programming languages such as GRAFCET, combined with Petri-net based formalism [13], to produce predictable behaviours. However, the ESPRIT vehicles main controller is an embedded computer system chosen for its ease of use in a prototyping process and its powerful Central Processing Unit (CPU) at the cost of a less predictable behaviour. On the final production vehicles, this computer will be replaced by a specific automotive controller.

The Esprit vehicles can be used in three different modes:

- Manual driving on the public roads.
- Autonomous "follower" mode for the towed vehicles.
- Autonomous parking, docking and un-docking mode when the road-train arrives in or leaves a station.

The first manual driving mode is similar to any current electric car: the user drives the ESPRIT vehicle with a steering wheel, a DNR gear speed selector (Drive, Neutral

and Reverse), an accelerator pedal and a brake pedal. The second mode is only used on towed vehicles to follow the front driven one and autonomously participate in propulsion, steering, braking with energy recovery and stabilizing the road-train on a single track path. The last driving mode is a fully autonomous mode when ESPRIT vehicles enter or leave a station in order to properly dock to the vehicles already at the station or un-dock with the vehicles to be left at the station.

B. *Contributions*

Since the ESPRIT vehicles will be driven by users on public roads, safety concerns have to be enforced. On the software point of view, the main controller runs a real-time Linux operating system with a fixed priority scheduler and a proprietary middleware helping with task management as well as providing a large amount of hardware drivers and communication facilities. Moreover, the use of a deterministic communication protocol, the Controller Area Network, between the vehicles allows us to use Worst-Case Execution Time computation tools [2] such as OTAWA [1], AiT [6], Bound-T [7], Chronos [9] or MBPTA [12] and validate the correct behaviour of the software architecture with Worst-Case Response Time (WCRT) analyses for fixed priority scheduling [10].

Contributions are the providing of a real-time architecture guaranteeing that the WCRT does not exceeds the specified deadlines not only for the software tasks involved but also for the end to end communication between modules.

This paper is organized as follows: first, the global communication architecture and the different modules involved in this process are introduced. Then the main controller is presented with its task-based, real-time operating system. In the last part, current results and possible improvements to create a more robust system are discussed.

II. Global architecture

The communication architecture of ESPRIT vehicles is separated in two main parts: the communication between the different modules inside one vehicle (intra-vehicle communication) and the communication between vehicles (inter-vehicle communication). In the first part of this section, the interactions between each controller and the developed communication architecture to make these interactions possible are explained. In the second part, the different hardware controllers embedded in each vehicle are described.

A. *Communication network*

Because ESPRIT vehicles can be driven in single car or in road-train configuration, the communication architecture involved needs to be safe and robust. For this architecture to work seamlessly in both configurations, internal communications are separated from the vehicle-to-vehicle data transfer.

1) Internal communication: ESPRIT vehicles use many off-the-shelf automotive components. So, to comply with the standards in the automotive industry, the Controller Area Network (CAN) protocol was selected for all communications between the car's modules (see [11] for an overview of vehicle networking using the CAN bus). The automotive industry is mainly using 500kbaud bus speed that has become a *de facto* standard among manufacturers. So, to comply with this standard, we also use 500kbaud bus speed.

The communication architecture was designed to cope with the large amount of data exchanged between the different controllers and between the different vehicles, since most of the CAN frames are periodically sent every 10ms. Moreover, the main controller (explained in section II-B4) is the central hub of all the internal communication as well as the vehicle-to-vehicle communication. So, we designed a "three and a half" CAN buses communication architecture, as shown in figure 2. A safe behaviour of the complete system needs to be ensured if one communication lane fails. For that, two partially redundant CAN buses are used for internal communication and two separate buses are used for

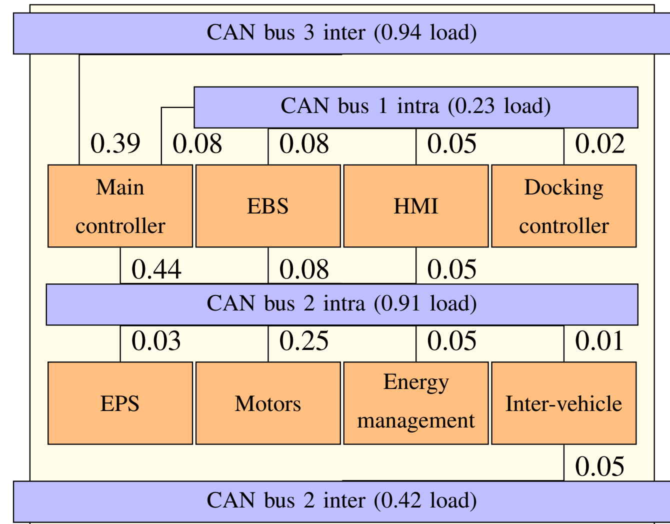


Fig. 2. The main communication architecture for one ESPRIT vehicle.

vehicle-to-vehicle communication, one of them being one of the internal buses but filtered to limit the bus load. The two internal buses, namely *CAN bus 1 intra* and *CAN bus 2 intra*, are only partially redundant due to hardware limitations on several modules that are only equipped with one CAN interface.

The use of *CAN bus 1 intra* and *CAN bus 2 intra* buses depicted figure 2 is done in redundancy whenever possible and more specifically between the main controller, the Electronic Brake System (EBS) and the user interface (HMI). Other modules are connected on the communication buses to balance bus load and maintain a safe data transmission, *CAN bus 1 intra* being exclusively an internal bus. The second CAN bus (*CAN bus 2 inter*) is also used to transfer low level data between vehicles, such as energy balancing in the charging process, when the main controller is powered off. Lastly, the third data line (*CAN bus 3 inter*) is used to exchange information between vehicles in road-train configuration.

Most of internal communication messages are exchanged on *CAN bus 2 intra*. *CAN bus 1 intra* carries 17 frames for a total bus load of 23%. *CAN bus 2 intra* transmits 74 frames cumulating at 91% bus load and it provides a 2 frames vehicle-to-vehicle communication participating in 5% of the 42% bus load on *CAN bus 2 inter*. *CAN bus 3 inter* allows a data transmission up to 38 vehicle-to-vehicle frames depending on the amount of vehicles in the road-train for a maximum bus load of 94%. Values on the right of the data lines in figure 2 are the CAN bus load emitted from each module. The bus load for *CAN bus 2 inter* and *CAN bus 3 inter* does not correspond to the connected module's emission load because it depends on the amount of connected vehicles and thus, it is the maximum possible load.

2) Vehicle-to-vehicle communication: The vehicle-to-vehicle communication serves two purposes: firstly it allows the main controllers to communicate for control purposes and secondly it permits a correct energy management when several vehicles are coupled.

For a correct behaviour of each vehicle in road-train configuration, an advanced adaptive and dynamic control law was developed. The control law role is twofold: it is designed to stabilize the road train in order to prevent lateral oscillations and it is guiding the road-train on a mono-trace trajectory by adjusting the steering and wheel torques of each vehicle.

The control law needs to be computed on a single main controller, while requiring the knowledge of the complete state of the road-train. Thus, a master-slaves scheme is implemented among involved main controllers: main controller of the first vehicle in the road-train is the master controller collecting data from its own vehicle and from all connected slaves, in order to compute the relevant setpoints. These setpoints are then sent to slave controllers. Each slave controller is responsible for the dispatching of received setpoints to the relevant controllers. For example, if the master controller's control law computes a torque setpoint for a slave vehicle, the slave controller has to determine if the request is a motor torque or a braking torque and transfer the setpoint either to the Electronic Brake System or to the Motor Inverters.

However, our safety analysis showed that a single failure on the master main controller may lead to a catastrophic behaviour. To limit the risks, if the master controller fails one slave controller takes its role and keeps the road-train in a degraded safe mode where the driving capabilities are limited. The limited driving capabilities are a limited maximum speed, direct brake setpoints from the driven vehicle measured on the brake pedal and a limited road train stabilisation with reactive steering on slave vehicles.

The Energy Management controller role is to ensure a sufficient amount of energy for each vehicle to work properly. For that purpose, it has the possibility to balance energy between vehicles when needed. Moreover, when vehicles are powered off, the Energy Management controller remains in a low power ON state in order to manage the battery charging procedure when several vehicles are connected. The Inter-vehicle controller also remains powered ON to allow vehicle-to-vehicle communication on *CAN bus 2 inter*.

B. Hardware controllers

Each vehicle is equipped with eight controllers, each dedicated to a specific task. In the following, controllers are presented in order of criticality on a decreasing scale.

1) Braking system: The first and most critical module is the braking system. This module is an off-the-shelf Electronic Brake System (EBS). It is made of an hydraulic circuit and an embedded controller, as shown in figure 3.

The hydraulic circuit needs two pressure inputs directly connected to the brake pedal master cylinder. Then, it is redirecting the pressure to four brake callipers, one for each wheel of the vehicle. In addition to that hydraulic circuit, a custom-developed embedded software can also generate braking pressure for each brake by using a hydraulic pump. The embedded controller can also actuate the Electric Parking Brake (EPB).

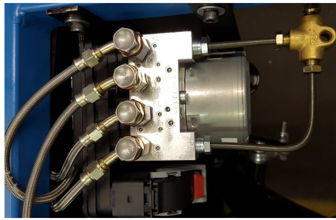


Fig. 3. Electronic Brake System installed in the first prototype.

Since the Electronic Brake System is a critical system of the vehicles, it is connected to two CAN buses (*CAN bus 1 intra* and *CAN bus 2 intra*) with a specific redundancy process: on normal operation, it is reading and writing on both buses and it is comparing the received values. If both values are identical, the communication is correct. However, if an error occurs on either CAN interface, it deactivates this CAN interface, relying only on the other one and it sends a warning status. If both CAN buses fail, the Electronic Brake System deactivates itself and works only as a pressure transfer module from the brake pedal to the brake callipers.

2) Steering system: The second most critical system is the vehicle steering. As for the EBS, an off-the-shelf rack-and-pinion Electric Power Steering (EPS), as shown in figure 4, that is used in current production cars was chosen. This module is using a standard software providing two operation modes: steering assistance and *City Park*.

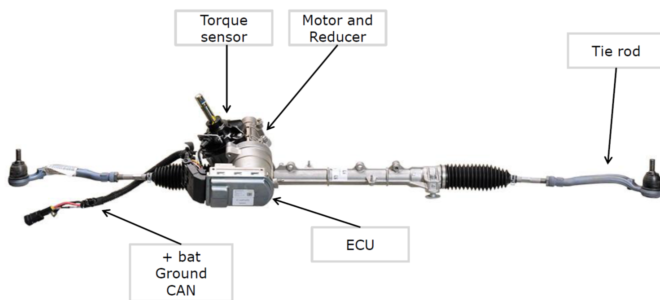


Fig. 4. Electric Power Steering used in the ESPRIT vehicles.

Steering assistance is done by accompanying the steering wheel movements with an electric motor using the embedded steering column torque sensor. The *city park* mode uses received setpoints on the embedded controller to actuate the steering rack. In this mode, the embedded steering column torque sensor is only used to check if a torque is applied to the steering wheel. It allows the user to stop the autonomous steering rack actuation by manipulating the steering wheel.

Even if this system is the second most critical in the vehicle, it only has one CAN bus interface. If the CAN bus fails, the electric motor power supply is automatically switched off. Nevertheless, the steering wheel can still be actuated by the user since it is directly connected to the rack-and-pinion.

3) Motor management: ESPRIT vehicles are equipped with two electric propulsion motors, one for each rear wheel. Each motor is driven by its own power inverter.

Inverters are equipped with an embedded controller that is managing motor speed and motor torque depending on the desired operation. It is also in charge of monitoring the input voltage and current coming from the main battery. Lastly, it allows the motors to act as generators to recharge the battery during motor braking.

On the vehicle's criticality scale, the motor management is the third most critical system but, as for the steering system, the power inverters have only one CAN bus interface. In case of failure of a CAN bus, motors power supply is switched off to let them in free rotation.

4) Main controller: The main ESPRIT vehicles controller is an embedded computer dedicated to four main tasks:

- Managing the three driving modes (manual, follower and docking) and allowing or not transitions between these modes, depending on conditions.
- Computing an advanced control law in road-train configuration, that is dedicated to road-train stabilization, path following and driving torque repartition.
- Collecting inputs from the user interface and transmit the data to the correct controller or module.
- Collecting and sending data from and to other vehicle's main controller.

The main controller implementation is further described in section III.

5) Inter-vehicle communication: As ESPRIT vehicles can be docked or un-docked to form a road-train, they need to be able to communicate together. This communication is done by a specific controller dedicated to messages filtering, messages redirection from one vehicle to another one and impedance adaptation on the data bus line.

One important aspect of this controller is its low data transmission time in order to minimize communication delays between vehicles. In the current implementation, this delay is about $260\mu s$ per controller.

6) User interface: The *user interface* (HMI) is made of several elements available to the driver: an accelerator

pedal, a DNR gear speed selector, a start/stop push button, a dashboard screen and a steering column switch cluster.

The accelerator pedal is providing an analogue signal proportional to the pedal position, the DNR selector and the start/stop button are providing digital signals. On the other hand, the switch cluster is using a low speed CAN bus that is incompatible with other buses used otherwise on the vehicle. In order to connect these devices to the existing buses, we are using a dedicated low level embedded controller for data conversion and further transmission.

Also, the dashboard screen is using its own controller to collect useful data to be displayed to the user, such as vehicle speed, odometer distance, DNR actual position, battery charge and driving mode.

7) Station docking controller: When an ESPRIT vehicle arrives into a station, its driving mode is switching to *docking*. In this mode, the main controller switches its inputs from the user to the station docking controller. Thus, the vehicle is completely autonomous but the user can still brake in case of emergency.

The station docking controller has three main roles: to localize the vehicle in the station, to compute a feasible trajectory to dock with other vehicles, and to operate the coupling device.

- The localization algorithm uses specific sensors such as laser and ultrasonic range-finders to accurately localize the vehicle in the station. The station is equipped with four *gates* indicating the station entry, the parking entry, the parking exit (which is also a charging station) and the station exit, as shown in figure 5. The localization algorithm then provides to the main controller two types of information: the exact position of the vehicle and between which gates the vehicle currently is.
- The trajectory computation algorithm uses the localization knowledge to determine a feasible path from the current vehicle position to either the parking exit if the station is empty or the already parked vehicle to dock with. This trajectory is then sent to the main controller to be followed autonomously.
- The docking coupler is an actively released and passively coupled and locked device. It has to be actuated in order to start the coupling process. Then, the station docking controller is releasing the coupling device when the vehicle is located close to the docked one.

8) Energy management: Since the ESPRIT vehicles are battery powered electric cars, a smart energy management system is embedded. It has three roles depending on the road-train configuration:

- It manages an efficient charging scheme when the vehicles are connected to the charging station.
- It limits the current available to all subsystems in driving mode depending on the battery charge.
- It shares and balances power between vehicles in road-train configuration.

III. Main controller software

The main controller is the brain of ESPRIT vehicles. It is concentrating most of the data exchanged between each subsystem.

In the software architecture design, it was first decided which *logical modules* are needed to perform all the main controller's work. These logical modules were then implemented on tasks of the embedded computer's middleware.

A. Logical modules

We identified four roles for the main controller in section II-B4. However the mapping of these roles onto logical software modules is not direct:

- The communications are exclusively done through three CAN buses so, we need a communication driver to handle all the data transfer.
- The control law for driving and stabilizing the road train is intended to be used in road train configuration only, so its execution depends on the driving mode state-machine's state.
- The three driving modes have to be selected depending on the road train configuration, the user inputs and whether the road train is in a station or not. We have decided to implement this module with a state-machine.
- Depending on the driving modes, the setpoints to the controllers does not come from the same inputs. For example: in manual mode, the user directly drives the vehicle's steering, motor and brakes but in follower mode, the control law drives the steering, motor and brakes depending on the user inputs. For this data switch to work as intended, we have implemented a data structure containing a data dispatcher.
- We also decided to implement an error management process running in background to check if the software, the communications and the hardware behave as intended during the vehicle's operation.

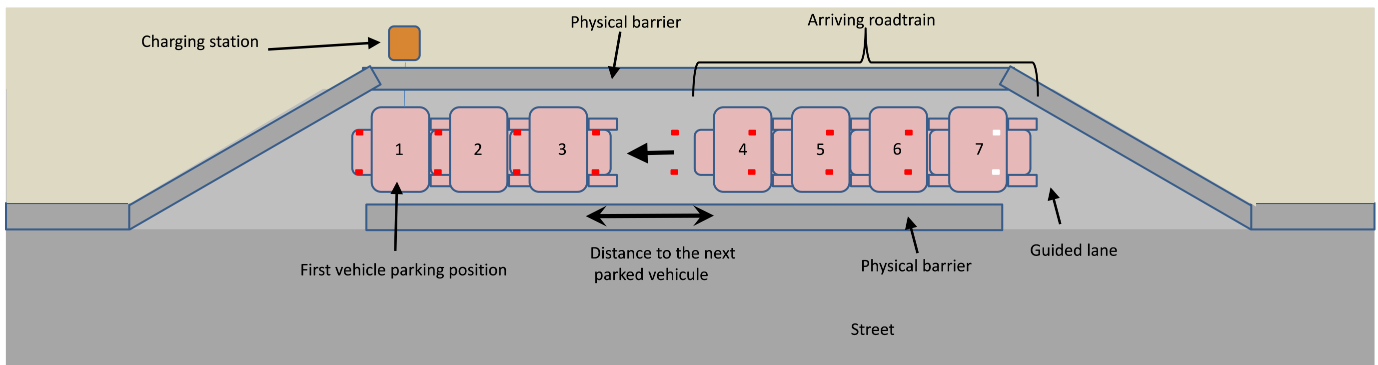


Fig. 5. Station conceptual view with an arriving road train being autonomously docked.

1) CAN communication driver: In the communication scheme, raw CAN2.0A [14] frames are used for data transfer across all sub-modules except for the motor inverters which use the J1939 protocol for CAN frame identification. Using raw CAN frames allows to specifically select the frame identifiers for each module and for each vehicle. The frame identifier selection is particularly useful in road train configuration since we are able to differentiate each vehicle depending on the CAN frame's identifiers.

In the CAN bus protocol, every module passively listens to any activity on the bus and can actively send asynchronous data. Our application needs to communicate with three CAN buses, so we require two software tasks per bus for an efficient data transfer: the first one is a periodic emission driver for sending data across the network. The second one is an event-based reception driver for collecting information from the other systems.

2) Control law: The control law used in the ESPRIT vehicles is designed around the kinematic and dynamic models of vehicles in different road-train configurations. It has three roles: firstly it stabilizes the road-train to avoid jackknifing and driving oscillations, secondly it allows the road-train to follow a mono-trace path, and thirdly it distributes driving power across the whole train.

For that, it needs input data from the master vehicle, such as its relative position, its speed and the steering input from the driver. It also needs the relative angles and speeds from all the follower vehicles. Then it computes the motor or braking torque for each wheel of the road-train and adjusts the steering of the follower vehicles.

Due to the highly dynamic nature of the road train configuration, this control law needs a 10ms computation period and a 20ms overall response time to achieve the best performance. The current control law manages to stabilize an eight vehicle road train in an elk test[1] at 45km/h.

3) Driving mode's state-machine: Depending on the situation, the main controller has to select one of these three driving modes:

- Manual driving when a user drives the car just as any regular electric vehicle.
- Follower mode when a car is towed and participates in the driving process.
- Autonomous docking when an ESPRIT vehicle or a road-train enters or leaves a station.

These three modes are organized as a state-machine for increased robustness. Each mode is further developed into sub state-machines to manage the specificities involved in each driving mode and transitions from one mode to another.

Moreover, the state machine embeds a *failsafe* state, in case an unrecoverable error occurs. This specific state sets the vehicles in a safe configuration with degraded dynamic performance and a limited maximum driving speed.

4) Data dispatcher: In order to correctly redirect data flows in vehicles, a dispatcher is implemented. It collects the input data from the user interaction, the control law computations and the vehicle-to-vehicle communications and redirects them to the actuators depending on the driving mode selected by the state-machine.

Five different paths were defined. They are specified from the vehicle configuration and the state-machine's mode: two paths exist when only one ESPRIT vehicle is driven and three additional paths are possible on road-train configuration.

- In single vehicle configuration:

[1] see ISO 3888-2 (https://www.iso.org/standard/57253.html)

  - Manual mode: data from the driver inputs are directly redirected to actuators, the accelerator pedal drives the motors, the brake pedal hydraulically actuates the brake callipers and the steering wheel moves the steering rack.
  - Autonomous docking: the docking controller computes the autonomous steering, motor torque and brake setpoints. However, for safety reasons, the driver keeps the braking capability with a direct actuation of the brake pedal.
- In road-train configuration:
  - Manual mode (only available for a master vehicle): data from driver inputs goes to the control law which computes the actuators setpoints for the motors and the brakes, steering is done by the driver.
  - Follower mode (only available for a slave vehicle): all actuators are driven by the master vehicle's control law through the slave's main controller.
  - Autonomous docking: same as in single vehicle configuration, except that the docking controller sends setpoints to the control law which in turn computes the desired actuator setpoints for all vehicles.

5) Error handler: In case of problem on any module within the road-train, errors need to be taken into account while still computing a safe behaviour. For instance, in case of a recoverable error, this process only warns the driver and quickly recovers to a normal behaviour. However, in case of an unrecoverable error it adapts the driving conditions by reducing the maximum speed allowed and asks the driver to stop the vehicle as soon as possible.

B. Task architecture

The logical modules presented above (section III-A) have to be executed on an embedded computer. The computer selected for ESPRIT vehicles prototypes is the Effibox Goliath 2000 (figure 6) from Effidence company[2]. It is equipped with quad-core Intel®Core™I5 clocked at 2.7GHz. This embedded computer runs a modified soft real-time version of the Ubuntu Linux operating system and a proprietary middleware [15].

1) Task model: The task model used in this embedded computer is defined by its priority, its period (for periodic tasks) and its CPU affinity. The tasks are scheduled using the fixed priority scheduler provided by the Linux kernel. However, the tasks are not synchronized by the operating system nor by the middleware so communication delays between tasks depends only on the task's periods.

[2]http://www.effidence.com

Fig. 6. Effibox Goliath 2000 embedded computer.

In the application, the most critical delay is the end-to-end response time from the data production to its use. For example, if the driver presses the accelerator pedal, the delay between the accelerator pedal position measurement and the motor actuation through the main controller has to be minimized, including all vehicle-to-vehicle communications.

In order to minimize this overall end-to-end data response time, the amount of tasks involved in any data path has to be reduced since the tasks cannot be synchronized. Moreover, the embedded computer's manufacturer, considering the used operating system and middleware, guarantees a correct behaviour of periodic tasks defined with a period larger or equal than 10ms. We then decided the fastest periodic task to be executed at a period of 10ms.

2) Logical module's implementation: With the task model imposed by the hardware and middleware, it was decided to implement the *logical modules* as follows:

- As stated in section III-A1, the three CAN buses needs six tasks to handle properly frame emission and reception. However, in our communication architecture, the *CAN bus 1 intra* and *CAN bus 2 intra* buses are partially redundant, so to maximize the data synchronization, we have grouped the emission processes for these two buses. Only five tasks have been implemented: two periodic ones for emission and three event-based ones for reception.
- One periodic task is dedicated to the error handler process to run in background.
- Another periodic task is dedicated to the state-machine, the control law and the dispatcher processes to minimize inter-task delays.

Figure 7 shows this repartition with the logical modules in green, the tasks in yellow and a shared data structure

in blue.

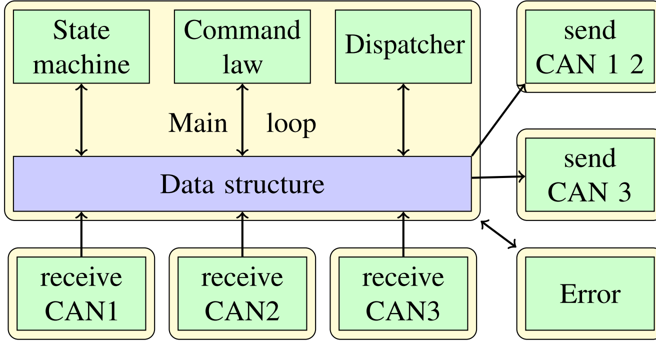


Fig. 7. Logical modules implementation on middleware's tasks and their data paths.

The task parameters for each task is described in table 1. In this implementation, all five tasks involved in CAN bus

task	period	affinity	priority
send CAN frames 1 2	10ms	core 1	medium
send CAN frames 3	10ms	core 1	medium
receive CAN 1 data	event	core 1	medium
receive CAN 2 data	event	core 1	medium
receive CAN 3 data	event	core 1	medium
Main loop	10ms	core 2	high
Error handling	100ms	core 3	low

Table 1. Tasks involved in the software architecture and their parameters.

management are dedicated to a single CPU core whereas the other two tasks run on dedicated cores.

3) Measured response times: To be able to analyse the software's execution and its response times, the data logging features provided by the middleware are used to trace the response times of each task.

The figure 8 shows three different response times measures in three different road-train configurations: 8a represents the operation of a single ESPRIT vehicle, whereas 8b and 8c shows respectively the response times of road-trains of two and eight vehicles. With this architecture, all the CAN frames do not need to be sent every 10ms, so to reduce the bus load, some CAN frames are sent on a lower frequency, so the *send CAN frames 1 2* (purple plot) and *send CAN frames 3* (green plot) have a large variation in response times. However, even if the *send CAN frames 1 2* maximum execution times are quite high (8ms) with relation to the task's period of 10ms, the CPU core dedicated to this task spends most of its time waiting for the data to be sent on the CAN buses.

In single vehicle operation, the only time consuming task is the emission of CAN frames on buses 1 and 2, all other tasks run under 0.5ms. However, in road-train configuration, the control law activates (cyan plot) and needs to transmit data on CAN bus 3. As we can see, all the tasks are able to run under their 10ms period.

On a data transfer point of view, when a CAN frame arrives on the bus, it wakes the corresponding receive task and at worst, it can be taken into account by the main loop 10ms later as shown in figure 9. The same delay (10ms) can be inferred to the periodic task non-synchronisation when a computed data is sent on the CAN bus, as shown in figure 10.

IV. Results

The main controller of the control architecture then has an impact of at worst 20ms between the arrival of an input data on the CAN bus and the output of the computed value. This delay gives a good behaviour in single vehicle configuration, when biggest delays come from the reactivity of the driver.

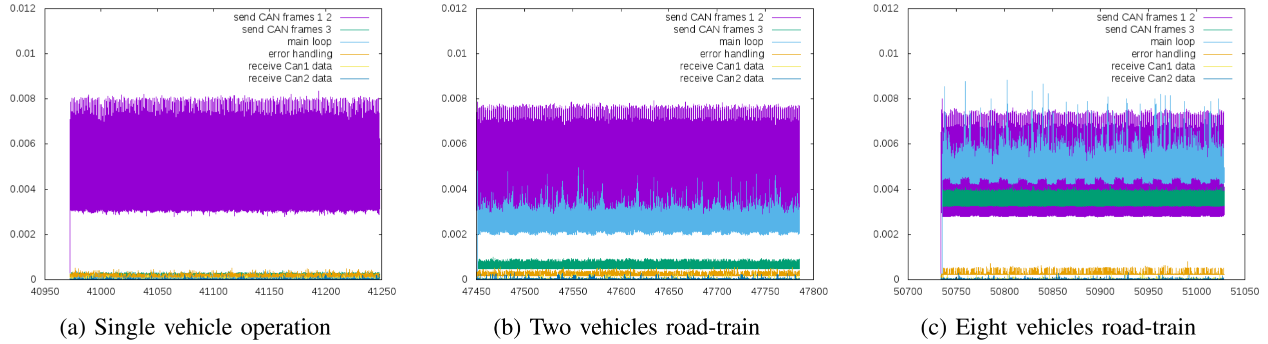
However in road-train configuration, since the control law is dynamically reacting to user inputs and road train configuration, the delay from a computed setpoint to the observation of its effect is critical for a correct operation.

Figure 11 shows the worst case delay for the data path from the main loop of the master vehicle's main controller to another controller on a slave vehicle and back. As we can see, this worst case delay adds up to 70ms. The dynamic nature of the system dictates that the stabilizing control law has to cope with these large delays in data transmission compared to single vehicle operation. As stated in section III-A2, the control law was designed to cope with 20ms delay between the computation of a setpoint to the observation of its effect in order to provide the best stabilization performances.

Moreover, added delays coming from the other modules than the main controller were not discussed in this paper. For example, motor inverters add a 40ms delay between a sent torque setpoint and a received torque change measure. The inverter manufacturer provided a prototype software to reduce the motor inverter delays. This new software is currently being tested to evaluate its performances.

In order to improve these response times, it is planned to investigate a better software management. These improvements are twofold:

- If the middleware manufacturer provides an update enabling task synchronization, we would be able to reduce the main controller delays from 20ms to 10ms and the vehicle-to-vehicle communication delays from 70ms to 40ms. Alternatively, we may



(a) Single vehicle operation (b) Two vehicles road-train (c) Eight vehicles road-train

Fig. 8. Measured response times in seconds for six of the tasks involved in the software architecture on a time-scale of five minutes.

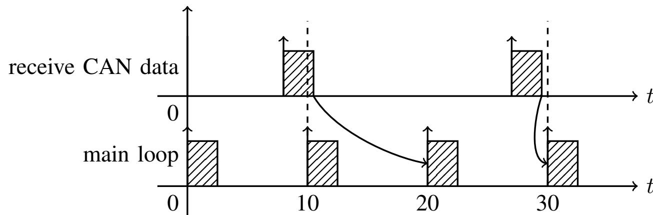


Fig. 9. Delays between data reception and possible usage.

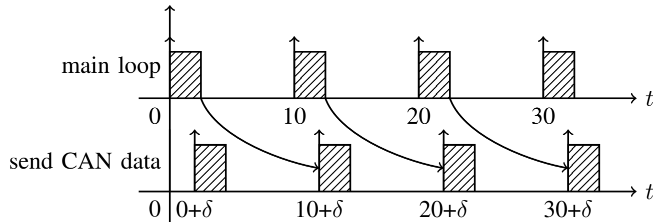


Fig. 10. Worst case delay between data computation and possible emission due to non-synchronous tasks.

have to use a different middleware implementing a task synchronization protocol, such as RT-MAPS [8].

- If this task synchronization is not possible with the current embedded controller, we want to optimize the task allocation to further reduce the amount of tasks involved in the application and to achieve a similar delay improvement as the task synchronization. However, we cannot merge some of the current tasks due to their Worst Case Execution Times.

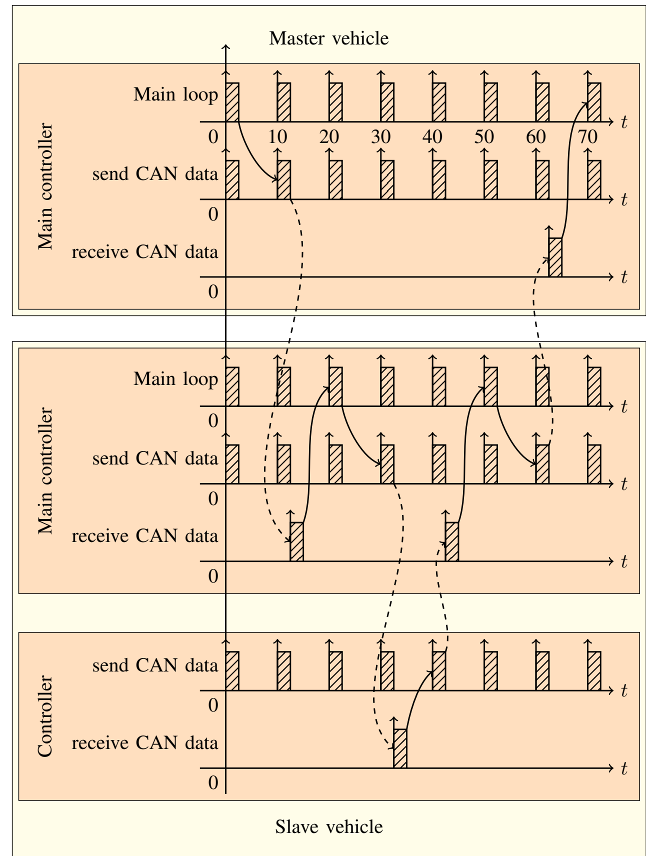


Fig. 11. Worst case delay between a computed setpoint and the measured effect through an inter vehicle communication ; Dashed lines represents vehicle-to-vehicle communications.

V. Conclusion

A communication and control architecture was designed for the navigation of a fleet of coupled vehicles. In this specific system, modularity and time performances are critical for a safe behaviour of the road-train. Design choices were guided by the selected hardware and real-time performances of the complete system.

At the moment, multi-vehicle communications have been tested on two real vehicles and up to eight simulated vehicles. Promising results were already achieved with a

single vehicle configuration and with a simulated road-train by connecting four real controllers together.

For future improvements, real road-train tests are planned to be performed and compared to simulations. End-to-end response times for inter-vehicle communications should also be reduced.

Acknowledgements

This research and development work was carried out in the scope of the Easily diStributed Personal RapId Transit (ESPRIT) project. This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement N°653395.

References

[1] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In *Software Technologies for Embedded and Ubiquitous Systems*, Lecture Notes in Computer Science, pages 35–46. Springer, Berlin, Heidelberg, oct 2010.

[2] I. Broster and G. Bernat andA. Burns. Weakly hard real-time constraints on controller area network. In *14th Euromicro Conference on Real-Time Systems*, 06 2002.

[3] Valery Cervantes, Peter Davidson, and Helen Porter. Developing a new mobility as a service concept. In *Association for European Transport*, 2017.

[4] L. Cheng and Y. Xu. Design of intelligent control system for electric vehicle road train. In *Proceedings of the 10th World Congress on Intelligent Control and Automation*, pages 3958–3961, jul 2012.

[5] ESPRIT. Easily diStributed Personal RapId Transit project, 2017.

[6] C. Ferdinand. Worst case execution time prediction by static program analysis. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 125–, apr 2004.

[7] N. Holsti, T. Lngbacka, and S. Saarinen. Worst-case execution time analysis for digital signal processors. In *2000 10th European Signal Processing Conference*, pages 1–4, sep 2000.

[8] Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *30th IEEE Real-Time Systems Symposium*, 2009.

[9] Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. In *Science of Computer Programming*, 2007.

[10] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, jan 1973.

[11] B.K. Ramesh and K. Srirama Murthy. In-vehicle networking. In *SAE Technical Paper*. The Automotive Research Association of India, 01 2004.

[12] L. Santinelli, F. Guet, and J. Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 199–208, apr 2017.

[13] M. Sogbohossou and A. Vianou. Formal Modeling of Grafcets With Time Petri Nets. *IEEE Transactions on Control Systems Technology*, 23(5):1978–1985, sep 2015.

[14] Technical Committee: ISO/TC 22/SC 31. ISO 11898 – Road vehicles – Controller Area Network (CAN). In *Car informatics. On board computer systems.*, 2015.

[15] C. Tessier, C. Cariou, C. Debain, F. Chausse, R. Chapuis, and C. Rousset. A real-time, multi-sensor architecture for fusion of delayed observations: application to vehicle localization. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1316–1321, September 2006.