Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Distributed Object-based Architecture for Controlling Autonomous Vehicles

Ronaldo Ap. Silva * **Julio E. N. Rico** * **Leandro B. Becker** *
Christian Kelber **

* *Department of Automation and Systems, Federal University of Santa Catarina, Florianopolis, Brazil, (e-mail: ronaldo@das.ufsc.br, julio@das.ufsc.br, lbecker@das.ufsc.br).*
** *Department of Electrical Engineering, UNISINOS, Sao Leopoldo, Brazil, (email: kelber@eletrica.unisinos.br)*

Abstract: In autonomous vehicles control, there is a need to interconnect several electronic component units (ECUs) in a proper manner to allow the embedded computational system to perform its control task in an efficient and reliable way. Therefore, this paper presents an object-based communication architecture that provides an infrastructure to design such systems. The proposed architecture relies in the publisher/subscriber protocol, which is characterized by the uncoupling between its components, anonymous communication with content-based messages, and many-to-many communications. Our architecture is based in the concept of sentient-objects, which are reactive entities distributed across the network that normally represent elements from the problem-domain. This paper presents the architecture, as well as an application used for controlling an autonomous vehicle.

1. INTRODUCTION

Technological advances spread the use of mobile robots in several application fields such as dust suction, mailing delivery, interplanetary and underwater exploration, etc. Given the large application spectrum, several distinct methodologies were developed to handle the problems of modeling, designing, and controlling such specific systems. One application of special interest in the scope of this work is the autonomous vehicles navigation (AVN) AMIDI [1990], OLLERO and AMIDI [1991], MANIGEL and LEONHARD [1992], SHNITZ [2001], WIT et al. [2004], JASCHKE [2002]. It can be used for different purposes including increasing comfort, optimizing gas consumption and pollution emission, and, the most important, enhancing safety MANIGEL and LEONHARD [1992], WIT et al. [2004].

In autonomous vehicles control there is a need to interconnect several electronic component units (ECUs) in a proper manner to allow the embedded computational system to perform its control task in an efficient and reliable way. The automobilist industry has widely adopted the use of fieldbusses like CAN-bus (BOSCH [1991]) and FlexRay (FLEXRAY [2005]) to interconnect the ECUs. However, the increasing complexity of such systems requires not only a physical interconnection among the ECUs, but also a logical one, which must be able to represent properly the interaction model among the components.

In BECKER et al. [2001] it is presented a study that compares different interaction models among distributed ECUs that are used to control a robot arm. Obtained results indicate that the Publisher/Subscriber (P/S) model is suitable to represent systems whose components are loosely coupled, that use anonymous communications in a many-to-many way, and that differ messages by contents. Such characteristics are also observed in the control system considered in this work, allowing us to conclude that P/S is a natural choice for the proposed communication architecture.

Despite a proper communication model, an architecture for autonomous vehicle control must be able to cope with quality-of-service (QoS) requirements, in special time constraints. Such applications require decisions to be performed in a timely manner, requiring real-time guarantees from the underlying infrastructure.

KAISER et al. [2005] introduced the concept of sentient-objects, which are autonomous entities that interact among each other using the P/S model under timing constraints. To support this concept authors present the COSMIC (COoperating SMart devICes) middleware, which represent an abstraction layer capable to support the design of applications using sentient objects on top of different networks, such as the CAN-bus.

This paper presents the redesign of the control architecture used in an autonomous vehicle, which was originally constituted by a network of PLCs (Programmable Logic Controllers). The new architecture presented in this paper substitutes the PLCs for low-cost computational nodes. The CAN-bus fieldbus is used to interconnect these nodes, instead of the proprietary protocol from the PLC manufacturer (ALNET BUS) as shown in Fig. 1. Moreover, the main enhancement introduced in the proposed architecture was the use of the sentient-objects concept, providing a loosely coupled distributed control architecture.

The rest of the paper is divided as follows: section 2 presents the vehicle used in this study, and also describes its control system; section 3 describes the middleware

COSMIC used to support the sentient-objects; section 4 presents the new distributed architecture designed to control the vehicle; finally, section 5 presents the conclusions of the paper.

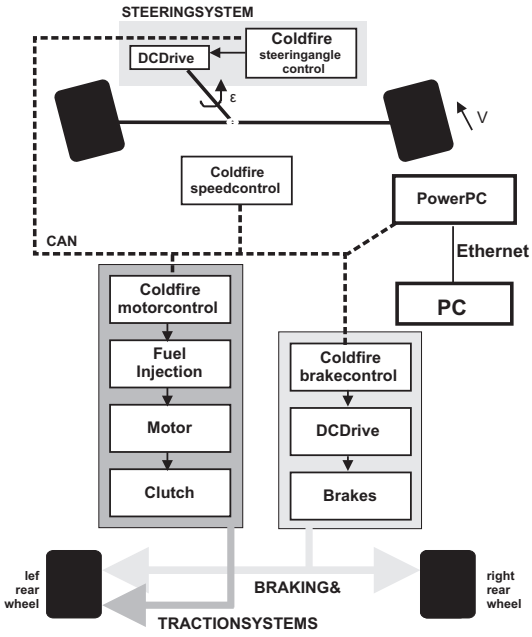

Fig. 1. Original Control System Architecture.

2. AUTONOMOUS VEHICLE CONTROL

The vehicle used in this study is a Mini-baja prototype, which was automated and converted into a drive-by-wire vehicle. This rear wheel drive vehicle weights about 200kg and is equipped with a 5HP internal combustion engine. In its original condition it could be guided like a conventional car and the maximum steering angle is about 0,79rad. The original control architecture was organized in different subsystems, being responsible for traction, acceleration, brakes and steering angle control.

Electrical drives were therefore used to control the throttle angle, the braking force and the steering wheel angular position. The drive-by-wire structure originally implemented in the vehicle was based on an ALTUS PLC network, with industrial PLCs interconnected to each other by an ALNET1-Bus. There is one PLC responsible for each subsystem, such as in a distributed control structure. Another two PLCs were used two for speed control and network management. The steering, traction and braking systems were analyzed separately and for each of them a specific controller was developed. For speed control an Adaptive Cruise Control (ACC) was implemented. The ACC system was separated into two branches, one for accelerating and the other for braking the car. Each branch presents some particularities, which were integrated in the vehicle's mathematical model. The speed controller has to accelerate, decelerate, and brake the vehicle, switching between both accelerating and braking branches. Both branches must not interfere each other, so a particular algorithm was designed to operate the switch.

Autonomous vehicle controllers are typically organized in cascade, as depicted in Fig. 2 (WIT et al. [2004], JUNG et al. [2005]). At the highest level (level 4) one can find the navigation planning and the trajectory generation. Path tracking control algorithms based on kinematics models are normally located at level 3, while dynamic control is at level 2. Finally, sensor/actuator control systems are at level 1. The original control system for the Mini-baja was organized in 9 computational tasks, as follows:

- *T_1* - Kinematics control loop;
- *J_2* - System's model and control law definition;
- *J_3* - Reference path definition;
- *T_4* - Integration error correction;
- *J_5* - Data transmission;
- *J_6* - Start application;
- *J_7* - Stop application;
- *T_8* - beta and theta Dynamic control loop;
- *T_9* - Speed dynamic control loop.

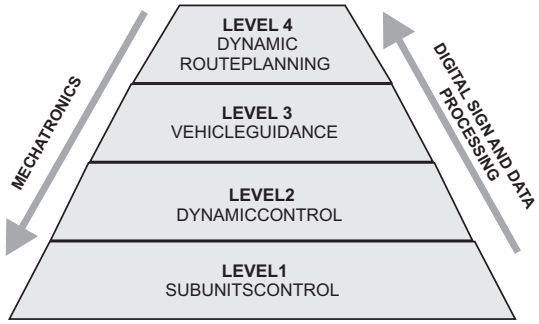

Fig. 2. Cascade-based autonomous vehicle control.

Tasks denoted with T are periodic and tasks denoted with J are aperiodic, which means that they do not occur within a fixed interval. Among these tasks, *T_1*, *J_2*, *T_4*, *T_8* and *T_9* should run concurrently while the vehicle is moving. *J_3* and *J_5* are executed only when the vehicle is stopped. *T_1*, *T_7* and *T_8* have hard deadlines, and that can never be violated, while the others tasks have soft deadlines, where violations are tolerated. For example, the soft condition of *J_2* is based on the pre-condition that the system will never require any parameters redefinition to treat a critical situation. That is, model and control law redefinitions during the driving process are assumed to be necessary for performance optimization, and not for emergency purposes. Similarly, *T_4* is defined as soft assuming that the periodic integrating error correction will never let this error accumulate up to critical levels. Finally, deadlines of the periodic tasks are assumed to be equal to the task's period.

Despite of the concurrency aspect, the original system implementation concentrated implementation from levels 2 to 4 on a single computational node (PLC), while level 1 was distributed among less powerful PLCs. In a first look, the available control system seamed to be distributed. However, it was designed in a tightly coupled

structure, despite of the fact that it is organized in different organization levels. Our proposal here aims to decouple the system both logically and physically. Here, logically decoupling means to make the system partitioning using the concept of sentient-objects. The physical decoupling is related to the deployment of the sentient-objects among low-cost microprocessed nodes, all interconnected by the CAN-bus fieldbus, as shown in Fig. 1. The concept of sentient-objects is supported by the COSMIC middleware, as described in the next section.

3. COSMIC MIDDLEWARE

The COSMIC (Cooperating SMart devICes) middleware from KAISER et al. [2005] was specially designed for embedded applications. It provides an API that abstract many low-level details from users. For instance, it hides the different addressing and routing mechanisms of various physical sub-networks. Moreover, it provides means to handle the different quality properties that underlying networks may have. Additionally, the middleware should support dynamic network configuration issues like changing or adding components without having to completely reconfigure all subnets.

Communications using COSMIC are content-based, following the Publisher/Subscriber protocol. Because the content or subject of a message becomes a major criterion for routing in such an interaction model, it is well suited as an overlay to mask the different addressing mechanisms in a system of interconnected heterogeneous subnets. The subject of a message has a meaning across multiple networks. This is exploited for filtering at the network boundaries. If a certain subject is not subscribed outside a specific subnet, it is not propagated by the respective gateway. Thanks to such approach, components can be removed, replaced or new ones can be added without involving an explicit address configuration effort in the components or in the application.

The second important point in a control network is the predictability of communication. Not all messages may need the same level of predictability and it is desirable to trade resource demands against predictability. COSMIC provides means to control this trade-off in an application-oriented way and particularly handle this for an end-to-end message transfer across multiple subnets.

To create applications using COSMIC one must use the API provided with the middleware. Every application *object* that communicate through the middleware must inherit from the *BasicObject* class. Thereby it can make full use of the P/S facilities. For instance, the *Speed* class from Fig. 3 represents a sentient-object that is allowed to make full use of the mentioned API. The <<BasicObject>> stereotype will be further used to represent elements using the API.

Objects communicating using COSMIC API follow a standard behaviour. To send a message the object creates an Event, which is further broadcasted in the related channel using the *publish* method. This happens by means of sending the event from the object to the *Event Channel Handler* (ECH), which is in charge of sending the event to all its subscribers. Events reception can happen either

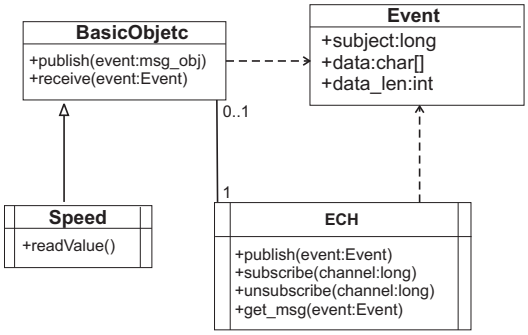

Fig. 3. Structure of the COSMIC API.

by polling or by automatic notification. As mentioned, all these communication steps follow a standard sequence, as in Fig. 4. The subscriber remains blocked in a illustrated loop waiting for the message reception. As the message is received, the related action is activated. Although a single subscriber is shown in the figure, such behaviour would be repeated for as many subscribers as were assigned for the message.

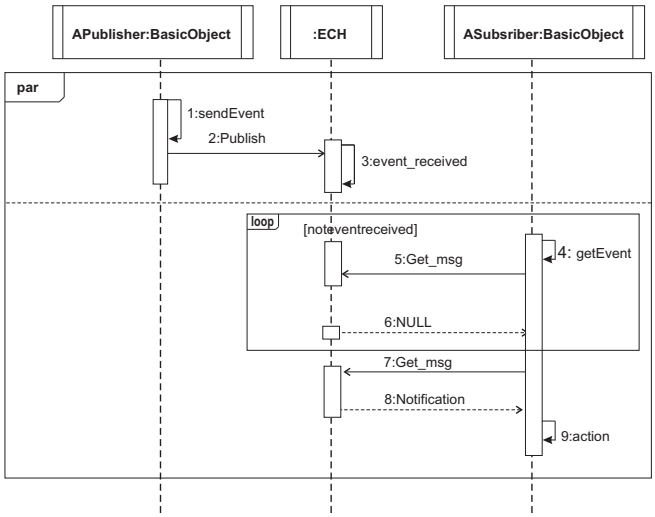

Fig. 4. Standard object interaction using COSMIC.

It is important to highlight the concurrent nature of the communication sequence presented in Fig. 4. Sending and receiving messages are actions peformed in different time instants. This is expressed by the *tag* par in the diagram. Another point to be mentioned relates to high abstraction level provided by the API, hiding many low-level implementation details from designers.

Next section presents the use of COSMIC and its API to design the new distributed control architecture for the Mini-baja vehicle.

4. THE PROPOSED ARCHITECTURE

This section introduces the distributed object-based architecture designed to control the autonomous vehicle described in section 2. The first modifications introduced in the original architecture were the substitution of the PLCs used in level 1 by microprocessed nodes based in

the Coldfire architecture. Additionally, the main control PLC was substituted by a PowerPC node. Moreover, a PC was used here to emulate a situation where cars could communicate with each other by using a wireless network. Given that COSMIC did not support wireless networks during the development of this work, we used an Ethernet connection instead. It is used to connect the PC to the PowerPC node.

Given the high complexity of the entire control system, we focused our description in the steering angle subsystem. Therefore we assumed that the vehicle moves with a constant and pre-defined speed.

As described in the previous section, all interactions within COSMIC are event-based. Such events are generated by the information producers, also known as publishers, which send their events using channels. Fig. 5 depicts the channel structure designed for the proposed architecture. The elements in the middle are the channels itself, while those elements surrounding the channels are the packages containing the objects.

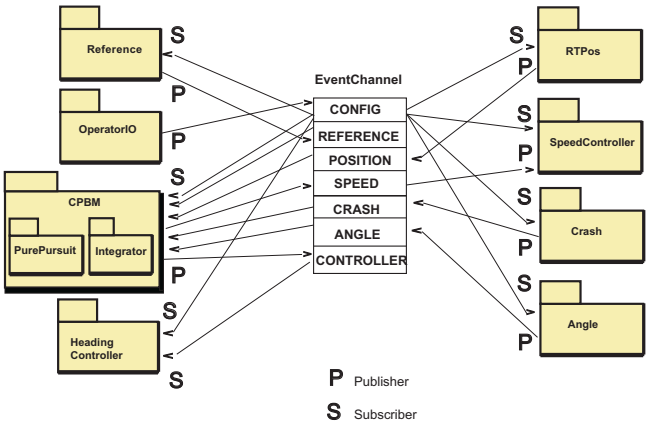

Fig. 5. Event Channel configuration.

We used a UML (OMG [2002]) deployment diagram to illustrate the computational nodes present in the designed application, as shown in Fig. 6. This diagram also depicts the network connections among the involved nodes, as well as the software components deployed in each node. It is important to highlight that each node contains an ECH component, offering an interface for the P/S protocol, which also means that COSMIC is active in the node.

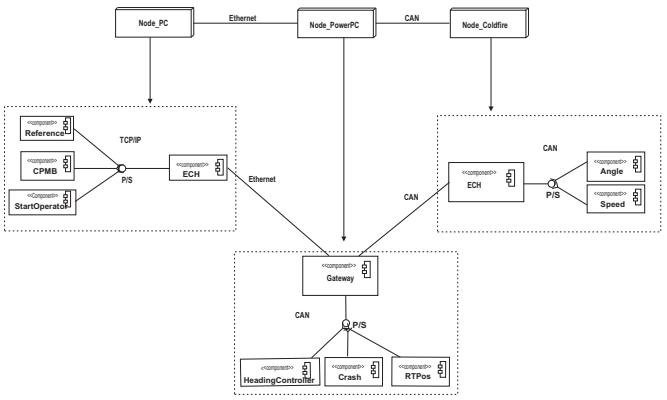

Fig. 6. Software components deployment.

Another important aspect from the diagram in Fig. 6 is the *Gateway* component in the PowerPC node. This component is in charge of providing the adaptation of messages exchanged among different networks, in this case Ehternet and CAN-bus. It also includes responsibilities from the ECH component, therefore a single *Gateway* component is enough to allow COSMIC to properly run in the node.

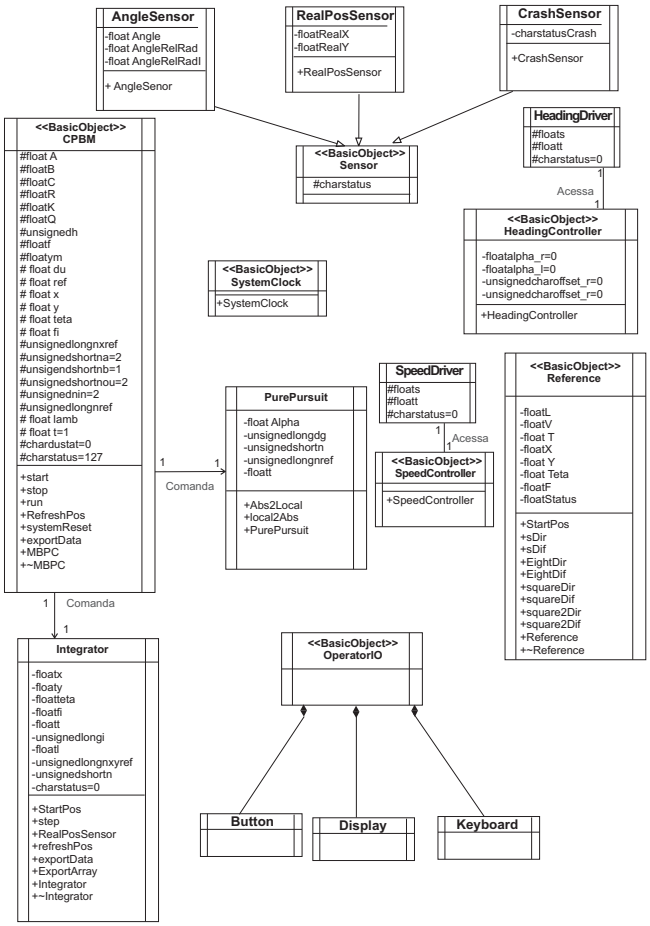

Fig. 7. Class diagram of the designed system.

The class diagram of the proposed architecture is presented in Fig. 7. Special attention should be given to the use of the <<BasicObject>> stereotype. All classes that contain this stereotype (either directly of from inheritance) are able to communicate using the P/S protocol by means of the COSMIC API. Follows a brief description of such classes:

- **CPBM**: implements the control algorithm (a Predictive Based Controller);
- **Reference**: is in charge of generating the reference trajectory;
- **PurePursuit**: implements the pure pursuit algorithm used to create the approximation trajectory;
- **Integrator**: implements the method in charge of determining, in real-time, the vehicle position in cartesian coordinates using numerical integration tech-

niques;

- **AngleSensor**: represents the electronic compass;
- **RealPosSensor**: represents the encoder;
- **SpeedController**: represents the speed control;
- **CrashSensor**: represents the anti-colision sensor;
- **SystemClock**: represents the real-time clock;
- **HeadingController**: represents the control actions sent to the steering angle control system;
- **OperatorIO**: represents the operator interface;
- **Driver**: contain parameters and characteristics of the hardware drivers.

Fig. 8 illustrates two common interactions in the designed system. The figure on left depicts the generation of the angle information. It is constituted by the communication between the *AngleSensor* object with the *SensorDevice* object. While the former represents the angle information obtained from the electronic compass, the latter is in charge of gathering the raw data from the compass. At every period t, the sensor data is gathered from the physical system, requiring the communication between *AngleSensor* and *SensorDevice* objects, which is done by method invocation. However, communication among the *AngleSensor* and the rest of the system is done by using the P/S protocol. The figure on right depicts the consumption of the angle information. The CPBM object, which is interested in the information, is notified by the middleware on the event occurrence. In this moment, it runs again the control algorithm and recalculation of the actuation information, which is further published to the interested nodes (in this case the steering angle control subsystem).

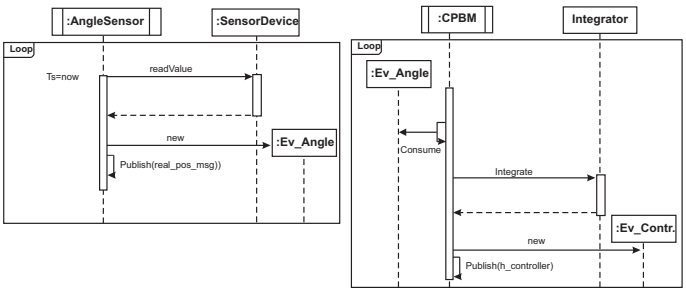

Fig. 8. Typical interaction scenarios in the designed system.

4.1 Evaluation Results

This subsection presents an evaluation of the distributed control system architecture presented in the paper. This evaluation consisted in observing if the system would follow properly the established trajectory. In Fig. 9 we present the graph from one of such experiments. The evaluated trajectory consisted of a S circuit, where the initial position of the reference trajectory (continuous line in the graph) is located at coordinate (1,1) in the (x,y) plan. The curves were designed to bound the range to 10 meters, strait paths are no longer than 20 meters, and the number of S is 3. The vehicle intentionally starts the trajectory in a different coordinate (-1,-5) (see the doted line), in order to proof that the control algorithm is able to forward the vehicle to the correct track.

The graph from Fig. 9 also shows the satisfactory behaviour of the vehicle while following the reference trajectory, given that the observed deviations are acceptable in this kind of control system. This leads to the conclusion that possible delays introduced by the proposed architecture (network and processing latencies) did not affect the overall performance of the system. Therefore, it is possible to assert that the proposed architecture is well adapted for controlling autonomous vehicles.

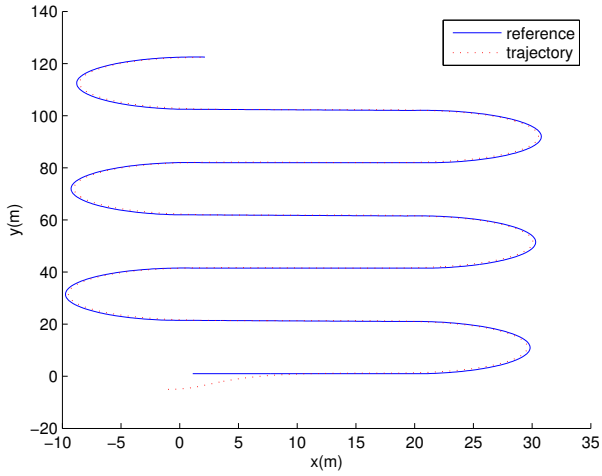

Fig. 9. Reference vs. performed trajectory.

4.2 Discussion

The original control system presented in section 2 was very dependent of a given hardware platform and communication network. There are several drawbacks of such approach, including a high dependence from a specific hardware manufacturer, and the high cost from PLCs in comparison to standard microprocessed nodes commonly used by automobiles manufacturers. Although the original architecture was very robust and reliable, it would never be adopted by the industry due to the just mentioned drawbacks. To overcome these problems, the new architecture presented in this paper has used low-cost microprocessed node, and a standard communication bus, highly used by the industry.

Analyzing the previous software structure, one could observe a tight coupling among all components. The definition of different layers in the adopted controllers was more used for documentation purposes than to improve the quality of the designed application. However, in the presented architecture all software components are loosely coupled.

From the designer perspective, the new architecture provides a huge enhancement by avoiding dealing with specific platform details, given that they are treated in the middleware level and not by the application. Moreover, it pro-

vides a much more natural way of structuring applications, focusing objects interrelation in the shared information, and not in the required object references to allow gathering such information.

5. CONCLUSIONS

This work concerned the study, design, and implementation of a distributed object-based architecture used to support the control system used in autonomous vehicles. This architecture is based in the Publisher/Subscriber (P/S) protocol, and is supported by the use of the COSMIC middleware. Such middleware allows software components to communicate using distinct network mediums, such as CAN-bus and Ethernet.

The proposed architecture was presented here by means of UML diagrams, representing both the internal interaction from the middleware perspective and also from the application perspective. Experimental results show that the decoupled architecture generated is not affected by possible communication and processing delays. These results were obtained by using an autonomous vehicle in charge of following a given trajectory. The simplicity and small budget of the proposed architecture makes it a suitable choice for the automobilist industry in real projects.

Additionaly, this architecture would benefit future applications for autonomous vehicle navigation. This property can be easily obtained by changing the Ethernet connection for wireless communication.

REFERENCES

Omead AMIDI. Integrated Mobile Robot Control. Technical Report CMU-RI-TR-90-17, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 1990.

Leandro Buss BECKER, Carlos MITIDIERI, Claudio VILLELA, Joerg KAISERr, and Carlos Eduardo PEREIRA. On evaluating interaction and communication schemes for automation applications based on real-time distributed objects. *Int. Symp. on Object-Oriented Real-Time Distributed Comp.*, pages 217–224, 2001.

Robert BOSCH. CAN Specification. Technical Report version 2.1, BOSCH, Stuttgart, 1991.

FLEXRAY. FlexRay Communications System Protocol Specification. Technical Report version 2.1, FlexRay Consortion, 2005.

K. P. JASCHKE. *Lenkregler fr Fahrzeuge mit hoher Schwerpunktlage*. PhD thesis, TU-Braunschweig, 2002. in german.

Claudio R. JUNG, Fernando Santos OSRIO, Christian R. KELBER, and Farlei HEINEN. *Jornada de Atualizao em Informtica JAI2005*, chapter Computao Embarcada: Projeto e Implementao de Veculos Autnomos Inteligentes, pages 1358–1406. SBC, 2005. in portuguese.

Jorg KAISER, Cristiano BRUDNA, and Carlos MITIDIERI. Cosmic: a real-time event-based middleware for the can-bus. *J. Syst. Softw.*, 77(1):27–36, 2005. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2003.12.037.

J. MANIGEL and W. LEONHARD. Vehicle Control by Computer Vision. In *IEEE Transactions on industrial electronics*, volume 39, pages 181–188, 1992.

A. OLLERO and O. AMIDI. Predictive Path Tracking of Mobile Robots: application to the CMU Navlab. In *Proc. of the IEEE International Conference on Advanced Robotics*, pages 1081–1086, Pisa, Italy, 1991.

OMG. UML Profile for Schedulability, Performance, and Time Specification. Technical Report OMG document n. ptc/, OMG, Seattle, 2002.

I. SHNITZ. Querregelung Eines Autonomen Strassenfahrzeugs. In *Forschritt-Berichte VDI*, volume 8, 2001. in german.

J. WIT, C. D. CRANE III, and D. Armstrong. Autonomous Ground Vehicle Path Tracking. *Journal of Robotic Systems*, 21(8):439–449, 2004.