Using COSMIC – A real world case study combining virtual and real sensors

Michael Schulze and Sebastian Zug
Otto-von-Guericke University
Faculty of Computer Science
Department of Distributed Systems (IVS)
Universitätsplatz 2
D-39106 Magdeburg
{mschulze, zug}@ivs.cs.uni-magdeburg.de

Abstract

The cooperation of distributed nodes in sensor networks forms a dynamic structure of information providers and information consumers termed as sources and sinks. Often, the used nodes differ by the available performance, network capabilities, operating system, applications etc. although, all of them have to be integrated in an appropriate network structure. Hence, a middleware is necessary to provide a common communication interface for the network in the whole system to cover the heterogeneity. To enable the integration on different platforms and into different systems the COSMIC middleware itself is designed flexibly and adaptively.

In this paper we present a cross platform case study, which shows the information exchange via COSMIC between micro-controllers and PCs on different network types by C or C++ applications and Matlab/Simulink. The case study illustrates, apart from other features, the possibility for an experimental setup combining virtual and real sensors/actuators in the sense of hardware in the loop scenarios.

1 Introduction

Complex mechatronic systems like cars, mobile platforms etc. join a large number of embedded sensor/actuator modules which individually or combined make information available, support calculations and data exchange or influence their environment actively. These systems are currently limited in their implementations and behaviour by a predefined structure of functionalities, interfaces and information sources. Additional interaction with other dynamically appearing components of an intelligent environment are neither intended nor possible. However, only if an application can use all available possibilities of its environment, its tasks will be optimally completed. Hence, for a distributed approach of data aggregation, analysis and the resulting interactions a middleware is necessary.

The role of the middleware in an embedded network is manifold: Firstly, as argued above, it has to hide the different addressing and routing mechanisms of the various physical sub-networks. All applications should use a common communication interface. Secondly, as the underlying networks often have different quality properties, it must provide means to handle this. Additionally, the middleware should support dynamic network configuration issues like adding or omitting components without reconfiguring all subnets completely. Therefore the chair of Embedded Systems and Operating Systems developed an event-based Publish/Subscribe middleware termed COSMIC (**CO**operating **SM**art dev**IC**es).

In this paper, we describe a case study based on an experimental setup using COSMIC to illustrate the potential of a common communication interface for the development of embedded systems. The paper is structured as follows: In section 2 we briefly introduce the concept of COSMIC. Section 3 describes the experimental platform, the application structure and analysis options and challenges. The conclusion and future remarks are summarised in section 4.

2 COSMIC

The COSMIC middleware described in [3, 4] offers an event-based communication model according to the publisher/subscriber concept. COSMIC is especially designed to allow cooperation between smart sensors and actuators on different hardware platforms ranging from 8-bit micro-controller up to 32-bit PC/Workstations and interacting over a broad variety of communication media like Controller Area Network (CAN) [8],ZigBee [9], TCP/IP to name a few.

In COSMIC an event is a programming abstraction and the carrier of the exchanged information. A COSMIC event consists of three different parts:

- a subject, represented by a unique identifier (UID) that describes the content,
- the content or payload itself for instance the value of a distance measurement and
- additional attributes (e.g. sensor position, context, quality) which are optional.

Events may arise in two different ways . Firstly, an event is spontaneously generated by the hardware because of a detection on a sensor interface. This means the physical environment is the stimulus of an event. Secondly, an event is periodically initiated by a clock to sense a change of a variable or of a state within the system.

Beside events COSMIC uses event channels as abstraction for event transfers. An event channel has the same subject as the corresponding event. The event is published by pushing it to the according event channel. In case of subscription, the occurrence of an event is notified to the application by the event channel. The programming abstraction event channel is introduced mainly to map the UID of an event to specific network addresses and therefore it hides the heterogeneity of the different network architectures by providing a global addressing scheme. Furthermore the event channel is used for network resource allocation - for instance a part of the bandwidth. Depending on temporal constraints or the importance of the event, the event is classified into three different quality levels which are hard real-time (HRT), soft real-time (SRT) and none real-time (NRT).

In COSMIC all events are handled by the event channel handler (ECH) which is part of the event layer (EL). The EL- marked by a filled circle in figure 1- is the interface to application level. Consequently, the publisher uses the EL to send events to event channels and on the other hand the EL provides the subscriber with notification and supports it by reading of events from event channels.

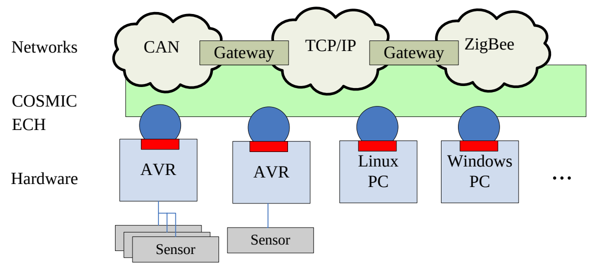

Figure 1. Current network structure

A possible implementation of the network structure with various hardware platforms as well as communication media is shown in figure 1. To allow information exchange between publisher and subscriber over network boundaries, the networks are connected by gateways. Currently, implementations of COSMIC for Atmel AVR micro-controllers, Motorola HC08, Siemens C167, Linux and Windows were realized which support the communication via CAN and TCP/IP.

The COSMIC middleware is implemented as a family [6] to achieve flexibility, adaptability and dependability. The implementation for the AVR microcontroller is an example of such a member of that family. The common functionality of the family is an intrinsic part of each member regardless of the platform where COSMIC is instantiated. Then the platform-specific functionality is only part of the relevant family member. Moreover, the design as a family allows a fine-grain selection of the required functionality necessary to spare the constrained resources on the embedded system. For example to deal with the limited memory (e.g. few kilobytes in sensor nodes), the software should only provide the functions actually needed by the application. In order to reach this goal, the functionality of the middleware is a collection of configurable components and functions. Design decisions about the required properties are deferred as long as possible and often determined by application needs.

Apart from the functional properties, the encapsulation of non-functional requirements - like dependability issues and real-time properties - need a separate treatment. These requirements - termed crosscutting concerns - are often fundamental system policies and refer to issues as robustness, fault-tolerance and real-time. Since crosscutting, quality features may apply to multiple functions and it is impossible to implement them as independent encapsulated entities. However, this would restrict the above-mentioned freedom of selection and adaptation. Aspect-oriented programming (AOP) seems to be a suitable possibility to deal with crosscutting concerns [5]. AOP allows separating the functional middleware components and non-functional components called aspects. Aspects are woven into the middleware during build time. Thus there is no extra runtime overhead to dynamically introduce these aspects.

3 Case Study: Interoperable network

3.1 Hardware description

As test environment for our software we use a development platform consisting of 4 nodes connected via a Controller Area Network (CAN). Each controller can be equipped with additional interfaces (sensor connection, ZigBee board, serial communication adapter) and is programmed separately or jointly using CAN very comfortably. One of the nodes includes a LED array for visualization. For simulation of failures, each controller can be switched off individually. Therefore, tests for a dynamically changeable network structure are possible. As depicted in figure 2, two sensors are integrated, i.e. a temperature and a distance sensor as representation of real sensor data. PCs can be integrated in the CAN network as well as being connected with each other via TCP/IP.

The micro-controllers run our PURE operating system and the PC works under Linux (in near future under Xenomai - areal-time Linux extension [1]) or Windows. The communication is handled by COSMIC.

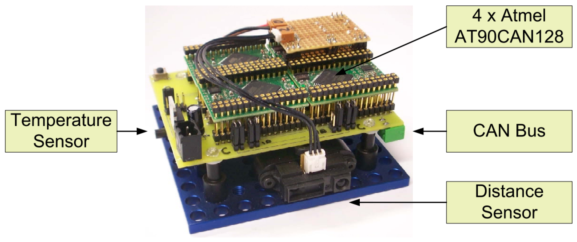

Figure 2. Development platform

As actuators, several mobile robots can be integrated in the network.

3.2 Software structure

The test environment demonstrates the flexibility of the distributed approach in two scenarios

1. "Real" sensors produce information which is consumed by other participants for data logging, visualization and data fusion.

2. Virtual sensors - like a PC - publish logged or calculated data instead of measured values.

The subscriber cannot differentiate between these two channel sources. Hence, the seamless interchangeability provides modern development methods like Hardware and Software in the Loop.

For the first case, our distance sensor periodically captures a voltage value as representation of the distance between sensor and an obstacle . This information is published on the CAN, and node 3 in figure 3 reflects the events by an LED array. The Linux PC works in two ways. Firstly, it acts as a gateway for the connection of CAN and TCP/IP. Secondly, a small C application runs, which is subscribed for the distance channel. It calculates a distance in $[cm]$ by the voltage value and logs all values at the same time. The use of Xenomai will offer real-time functionalities in this context. In Matlab/Simulink a median filter and a graphical user interface is used for a comfortable visualization of the sensor data. Further developments for data analysis, fusion and visualization will exploit the manifold number of toolboxes for Rapid Prototyping. In order to provide real-time applications such implemented models can be transferred and used by code generation tools for different platforms.

In the second scenario, sensor node 2 is switched off. Logged or calculated data are propagated by a Linux application or from Matlab/Simulink. Node 3 depicts the value as in the first scenario. This means if a mobile robot logs all information about its environment once, the replayed data can be used for instance in reproducible examinations of navigation or control algorithms .

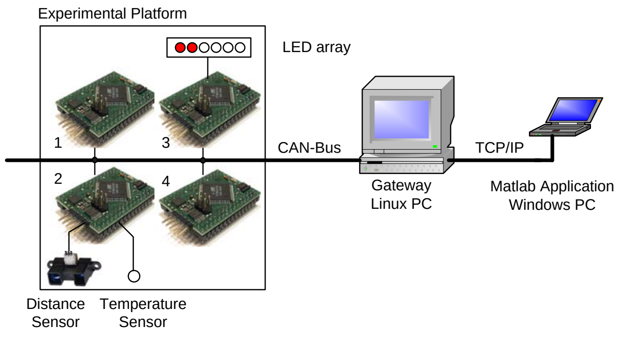

Figure 3. Scenario software scheme

4 Conclusion and Future Remarks

The simple demonstration setup illustrates the practical possibilities of a distributed system with a transparent communication via middleware.

1. A transparent and defined uniform interface supports application developers considerably. The design and implementation of distributed embedded systems can be accelerated by concentration on the application.

2. The dynamic adaptability of the system on runtime reacts to the appearance and disappearance of components caused by communication problems.

3. Exchangeable data sources and sinks simplify experiments combining real components and virtual modules. Hence, the test scenarios can use predefined reproducible information generated by a simulation first and validate the results by real measurement data.

4. The monitoring of the data transfer by established engineering programs supports Rapid Prototyping development for data aggregation, fusion and interpretation.

Point 1 in this enumeration is based on manifold versions of COSMIC. Hence, a family of COSMIC and appropriate operating systems are necessary for different hardware environments.

The current COSMIC implementation offers non-real-time communication only. For further extensions a time synchronisation will be included to offer real-time event channels. Therefore first drafts of the synchronisation algorithm presented in [2] reach average time deviations of $6\mu s$. This synchronisation is fundamental for more complex dynamic data fusion algorithms.

For an adaptability of the network mentioned in point 2 and 3 two aspects have to be considered. Firstly, each channel should provide information about its events like sensor type, measuring ranges, noise performance etc. Therefore [7] integrates functionalities for electronic data sheets and establishes appropriate service discovery functions in COSMIC. Secondly, intelligence for the information selection has to be designed for data sinks.

Point 4 is aimed at an advanced integration of Matlab/Simulink toolboxes for a flexible and dynamic data fusion.

As a more interactive test platform, we will use "Q" - a quad drive robot 4 - for further steps. Q represents the distributed approach very consequently. It consists of four independently steerable driving units, each monitored by a micro-controller and a number of infrared sensors. Additional gyroscopes and compass modules can be integrated. The network is connected by a CAN.

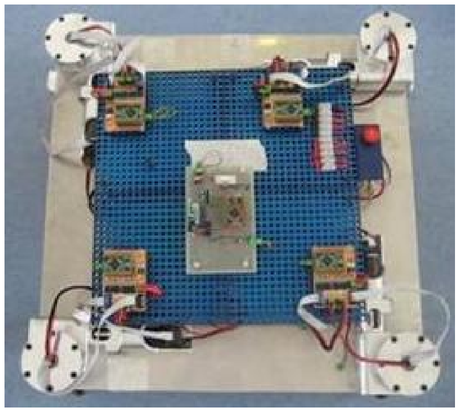

Figure 4. Mobile robot Q

The sophisticated mechanical drive system of Q offers movements with many degrees of freedom (e.g. moving and rotating simultaneously). Only with extensive communication, distributed controlling and flexible interaction of all modules tasks - "Driving through a door without collision" - can be completed. Using this in experimental setups, virtual sensors (smart bumpers, ultrasonic etc.) can simulate obstacles or disturbances for testing control algorithms. So the real environment with a robot system can be superimposed by virtual elements. After testing, the subscribers of the virtual environment are switched off and the robot works correctly without any changes in software. Complex moving patterns, path planning and so on can therefore be tested without the danger of physical damage or increased mechanical stress.

References

[1] Xenomai: Real-time framework for linux.

[2] M. Gergeleit and H. Streich. Implementing a distributed high-resolution real-time clock using the canbus. In *Proceedings of the 1st International CAN-Conference. Mainz, Germany*, 1994.

[3] J. Kaiser and C. Brudna. A publisher/subscriber architecture supporting interoperability of the can-bus and the internet. In *2002 IEEE International Workshop on Factory Communication Systems*, Väesteras, Schweden, August 28–30 2002.

[4] J. Kaiser, C. Brudna, C. Mitidieri, and C. Pereira. COSMIC: A middleware for event-based interaction on CAN. In *9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, Lisbon, Portugal, September 2003.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, June 1997.

[6] D. L. Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.

[7] H. Piontek and J. Kaiser. Self-describing devices in cosmic. 2004.

[8] Robert Bosch GmbH. *CAN Specification Version 2.0.* 1991.

[9] ZigBee Alliance. *ZigBee Specification - IEEE 802.15.4.* 2003.