Sentient Objects for Designing and Controlling Service Robots

Jörg Kaiser*, Sebastian Zug*, Michael Schulze*, Carlos Cardeira, Fernando Carreira*****

** Dept. Of Embedded and Operating Systems EOS, Otto-von-Guerike Universität Magdeburg, {kaiser,zug,mschulze}@ivc.cs.uni-magdeburg.de.*
*** Instituto Superior Técnico, Technical University of Lisbon,carlos.cardeira@ist.utl.pt*
**** Instituto Sup. de Engenharia de Lisboa, Polytechnic Institute of Lisbon, fcarreira@dem.isel.ipl.pt*

Abstract: Services related to healthcare and the support for elderly people become more and more important. Autonomous or semi-autonomous robots may play an important role in this area. From a control system point of view these robots are networks of distributed smart components to perceive their environment and react on it in real time. The problem of developing or extending such a robot often is that the designer has to start from scratch struggling with low level issues, where reusability of already designed components would be highly desirable. The paper describes a robot application in the area of a meals distribution service that combines two design worlds. One is the conventional world of modelling the functional properties without any structural considerations, the other is the world of cooperating sentient objects. We explain how the notion of sentient objects will assist the design, simulation and also later extensions and adaptations of the robot.

1. INTRODUCTION

The usual approach to the design and build a mobile robot includes algorithm design, simulations, virtual prototyping testing, etc. During the entire cycle, the necessary tools run in a centralized design system where all the information is present. Virtual reality toolboxes allow the visualization and validation of the proof of the concept. When dealing with reusability, flexibility and distribution, the conventional centralized approach is not appropriate. What is needed is an approach that provides a more flexible, and distributed way of implementation and also to define encapsulated reusable components that can be integrated in a new design. The paper introduces the notion of sentient objects which incorporate application functions and communicate via a well defined event interface. We also present an application scenario based on the I-Merc robot developed in a project dealing with robotic in health care. We will explain how sentient object will be used to allow reuse and the distribution of the processing tasks.

Application Scenario

One of the most important aspects inside hospitals is the meals distribution service and similar services since the quality of the food strongly influences the patients' recovery. Thus, extreme care is taken not only with regard to the food preparation but also to the transport between kitchen and patients. Normally, meals transportation is carried out by dedicated people that use trolleys specially devised for this service. A survey to some hospitals allowed us to conclude that the main drawbacks to traditional transport devices are their weight and the difficulty in handling them. Some of the more innovative health centres in the world started using mobile robots such as the HelpMate (Evans *et al.* 1992) or Transcar (Swisslog 2004) to transport different types of cargos between persons in services.

To increase the quality of the meals services inside health services, we proposed a more hygienic and efficient meal transport service, through a dedicated mobile robot, the i-MERC (Carreira et al. 2006). This robot is able to deliver personalized diets to patients and return to the washing room with soiled dishes to be carefully cleaned (see Figure 1). Although our application scenario is taken from the healthcare area, it can be extended to an industrial context where robots are used to carry semi finished products to the respective machines or deliver parts to respective stations.

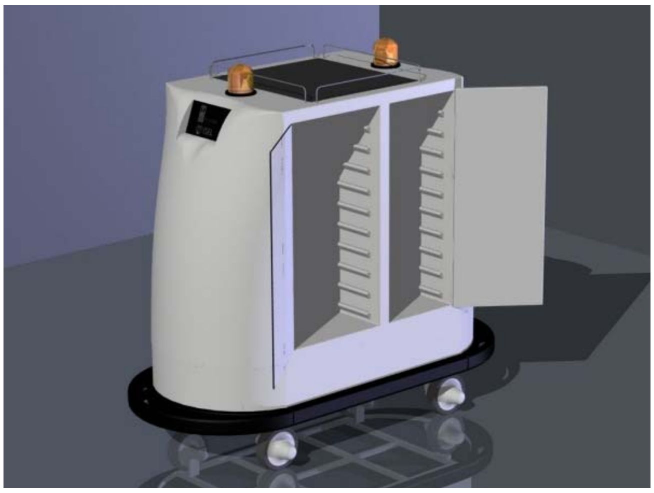

Fig. 1. I-Merc virtual prototype

2. Problem statement

Because of the diversity of applications, robot geometries and sensor and actuators requirements, every robot is developed from scratch modelling kinematics, specific physical sensor characteristics and developing the respective control programs usually on a very low level of abstraction. At the moment, there are two approaches to raise the level when modelling robot behaviour and derive the control program. On the one side there are approaches using Matlab/Simulink (Carreira et al. 2006) to define functional blocks and link these blocks to specify an overall robot behaviour. This is supported by a graphical interface and a rich library of functions eases the development of complex computational tasks like filters and evaluators. Additionally, sophisticated simulation capabilities enable early validation of the design. All these tools are based on a centralized model of control. The final generated code is monolithic and needs a powerful CPU to be executed. On the other side there is an emerging class of advanced mobile robotic systems which exploit the modularity and adaptability of tiny smart sensors and actuators to build the reactive layers of the system. With the availability of cheap computing and communication facilities like microcontrollers with integrated network access, such a component-oriented approach becomes attractive. The built-in computational facilities enable the implementation of a well-defined high-level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables. Consequently, the interfaces and the functions of these smart components may include functions related to overall control, supervision, and maintenance issues. The reactive level of a robot thus exhibits a modular system architecture in which smart autonomous components co-operate to control physical processes without the need of a central co-ordination facility. As it is further detailed below, we introduce the notion of a *sentient object* to model such components.

At the moment, the worlds of specifying control applications in Matlab/Simulink and the modelling of applications by sentient objects are pretty much separated. On the one side there are mature off-the-shelf development tools around Matlab/Simulink with the graphical programming tools, on the other side we observe an emerging world of co-operating smart components in which application code still is programmed on a rather low level. In most cases also low level communication primitives are available only because of efficiency and resource constraints. The problem currently is that the powerful modelling and simulation tools mentioned above cannot be exploited to reflect a distributed architecture of smart components. In the paper we will present an integration of these approaches. There are a number of benefits arising from this work. The designer can define identifiable, reusable components which have both, a representation in the system architecture with well defined interfaces and in the Matlab/Simulink world. There, the internal algorithms for the input-output behaviour can be specified and simulated. This enables the validation of such individual components before they are integrated into a larger system. Additionally, this approach facilitates simulation of complex systems by allowing flexible hardware- or software-in-the-loop techniques.

In the following chapters, we briefly introduce the sentient object model and the respective communication and interaction abstractions. Then we will describe how the simulation and a real robot interact and how sentient object will support the interaction between real and virtual components.

2. THE OBJECT AND INTERACTION MODEL

There have been many approaches for mastering the design process of complex artefacts by modularization and abstraction. Component-based and platform-based design concepts are well established examples (Szyperski 1998, Keutzer *et al.* 2000). Our approach follows the same goals providing encapsulated components with well-defined interfaces. What is highlighted in our concept is the autonomy of components and anonymity of communication that is inspired by the needs of large distributed sensor/actuator networks and supports easy composition and interaction (Kaiser and Mock 1999, Casimiro *et al.* 2004). We propose the notion of sentient objects to model the cooperating components in the distributed control system. The concept of a "sentient object" was proposed in the CORTEX project (Verissimo *et al.* 2002) and inspired by the work of Andy Hopper on sentient computing (Hopper, 2000) that deals with computations which are dependent on the physical environment in which they are performed. This is obviously true for an application as described above in which a robot interacts with its environment through sensors and actuators. The sentient object model and event-based communication are tightly related. Event-based communication (Bacon et al. 2000, Casimiro *et al.* 2004) reflects the spontaneous generation of messages and the respective notification of the receivers in a publish-subscribe style of interaction (Kaiser and Mock 1999). It should be noted that the term "event" does not refer to any synchronous (time-triggered) or asynchronous (occurrence-triggered) model of communication. Events are typed communication objects which can be disseminated in either way (Kaiser *et al.* 2001, Casimiro *et al.* 2004). The structure of an event is defined by the tuple <subject, attributes, contents>. The subject is related to the contents and indicates what type of information is carried in an event. The subject is used to route an event to the interested subscribers. A subscriber registers interest in a subject and is notified whenever a message with the respective subject is disseminated. The detailed description and discussion of the publish-subscribe approach can be found in (Kaiser *et al.* 2003). An important property is that a subscriber only has to specify the type of information it is interested in rather than the source from where this information has to be delivered. Vice-versa, the publisher does not have to address a certain receiver of an event explicitly. Therefore, a dynamic binding between publishers and subscribes is possible on the basis of subjects and is supported by the underlying COSMIC middleware. The attribute field describes context and quality attributes e.g. the time and location where such an event has been generated and how long an event will be valid. Finally, the content carries the payload data. An example of an event produced by a distance sensor in a mobile robot is given below.

distance_event := <UID, rel_pos., timestamp, validity, distance>

The UID identifies the subject of the event. "Relative position", "time stamp", and "validity" are attributes denoting the position where the distance measurement is performed (e.g. front or rear of a vehicle) and validity is a speed dependent attribute indicating how long this value is valid within defined error bounds. Finally "distance" is the actual measurement value. Subscribers may be a sentient motor control object to adapt the speed of the robot and/or an object which maintains map and position information. For a detailed description of the event-based COSMIC middleware, the reader is referred to (Kaiser *et al.* 2003).

Sentient objects have an interface that receives events, filters events according to their subject or attributes and autonomously decides about the actions to be performed and the events to be produced (Figure 2). The sentient object model reflects the needs of encapsulated autonomous components and mainly differs from the conventional object model in three respects:

1. The interface of a sentient object is defined in terms of events that are consumed rather than in terms of a signature defining the methods which can be performed by the object.
2. The reaction of an event is performed autonomously based on an evaluation of the event and its attributes. The attributes define aspects of the context in which such an object operates. This is in contrast to the invocation mechanism in object-oriented systems.
3. The interaction is spontaneous and in a producer-consumer style rather than in the client-server style of object invocations (Kaiser *et al.* 2001).

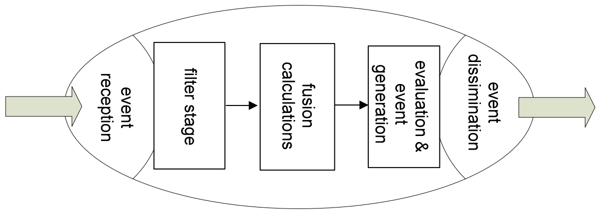

Fig. 2. Internal Structure of a Sentient Object

The internal structure reflects the processing of events. It is composed from a set of filters and processing components. The filter evaluates the subject and attribute fields of an event and selects those events which are of interest. The filter allows specifying events according to a certain location of origin or the time when they have been generated. The fusion part is the algorithmic component. Here, the incoming information is aggregated and combined with the information of other events. This part can be simple if only one type of event is considered, e.g. an average temperature is calculated from incoming temperature events or can be complex if many multi-modal sensor information coming from distributed sources have to be considered. The last component of a sentient object is the event generation part which disseminates the results of the evaluation process.

Smart sensors and actuators are a special form of sentient objects. Sensors perceive events from the real-world environment and producing respective (system-) events for the system's event layer. Vice versa an actuator consumes (system-) events and converts it to a real-world event by an actuation. Smart components therefore constitute the periphery, i.e. the real-world interface. The reception part of a smart sensor is an application dependent transducer and the output produced by an actuator are physical signals rather than system events. The sentient object model serves as a high level programming model defining the components and the interaction structure of the application. To program the internal sentient object structure, (Fitzpatrick et al., 2002) and later Biegel and Cahill (Biegel 2006) propose a rather heavy weight scheme where fusion is based on a Bayesian network and after identifying composite events, a rule-based inference scheme evaluates and converts them to output events. Furthermore, their model incorporates knowledge about the context in which the sentient object operates and adapts behaviour accordingly.

We propose to exploit the Matlab/Simulink programming and simulation environment to program the fusion and evaluation stages of the sentient object. Furthermore, Matlab/Simulink gives us the possibility to exploit hardware- and software-in-the-loop scenarios, i.e. enabling the interaction of real hardware components with simulated ones.

3. INTEGRATING THE OBJECT CONCEPT IN MATLAB/SIMULINK

Matlab/Simulink is a standard tool and provides high level programming and composition of functional blocks. Matlab/Simulink also allows specifying functional blocks for low-level communication. For the fusion and evaluation stages we can exploit the full properties and libraries of Matlab/Simulink to program the respective algorithms. Thus we are able to set up all components of a sentient object in a Matlab/Simulink programming environment. To support the use in the context of a sentient object, we added function blocks that realize the event-based communication. This allows distributing computations to multiple hosts and frees us from any low-level communication issues. The resulting blocks can be compiled to the native code of a target platform and thus we obtain a sentient object as an executable and reusable software component. Because of the event-based communication scheme, these components can be easily combined without changing any internal software e.g. configuring addresses or adapting to a specific low level communication protocol. Furthermore, the simulation facilities of Matlab/ Simulink enable the interaction of real and simulated sentient objects. Consider the small experimental setup depicted in Figure 3.

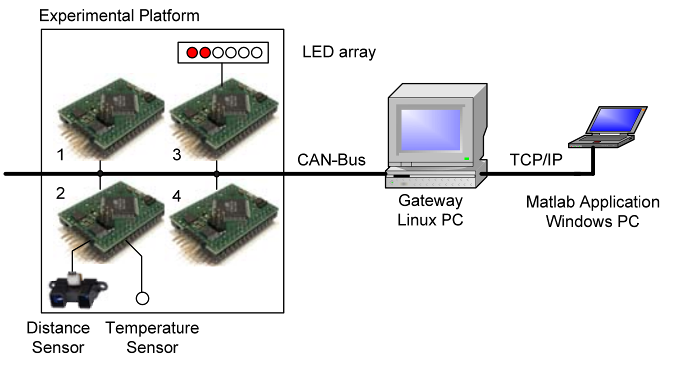

Fig. 3. Interoperability between simulated and real systems

Here a system of distributed smart sensors, actuators and tiny computing nodes (AT90CAN128) is connected to a PC running Matlab/Simulink. The different networks, i.e. CAN and a TCP/IP networks are connected. Because the COSMIC middleware is running throughout the system, sentient objects can transparently publish and subscribe events over network boundaries and gateways. This enables a number of variants between simulated and real sentient objects. Figure 4 sketches a simple application.

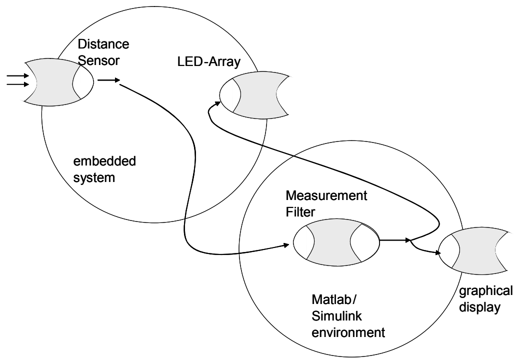

Fig. 4. Application objects

An infrared distance sensor should indicate the proximity of an obstacle. Because individual measurement results are not reliable, it will be processed by a complex filter. After filtering the information it will be send to a visualisation device. In the experimental hardware setup, this is just a line of LEDs. For filtering the sensor information, the "Measurement Filter" is encapsulated in a sentient object. It subscribes to the events of the distance sensor and provides a more reliable proximity information in its events than just taking the events from the sensor carrying the raw distance data. The visualisation object subscribes to this proximity event. In this scenario we now can mix real sensors and simulated ones. In a first stage the entire application may run on the PC under Matlab/Simulink. Because the components are sentient objects with the event interface, it is possible to use the real sensors instead of simulated sensors at later test and integration phases without major changes. In this way the parameters of the measurement filter can be adjusted before the respective object is compiled to the target hardware and migrated to one of the computing nodes. Any combination of simulated and directly executed objects is possible. Eventually, we can derive a standalone version with all objects running on the target platform. This enables an incremental design and implementation process and aids configuration tests considerably. The structure of the sentient object depicted in figure 2 is represented in Simulink as follow:

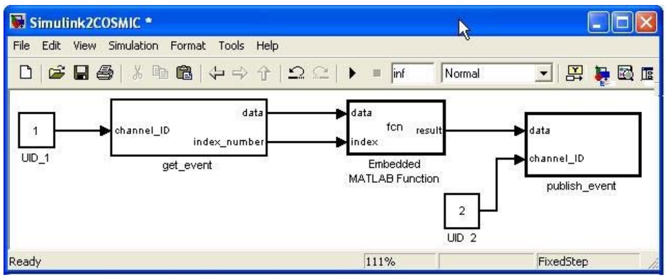

Fig. 5. Stucture of a sentient object in Simulink

The *get_event* block provides the current data of a channel indicated with *UID_1*. The *Embedded Matlab Function* block contains the three internal stages of a sentient objects: event filtering, data processing and event dissemination. According to figure 4 the data processing is used for a simple averaging of the last 10 incoming values. The results are published into the networks by the function block *publish_event.*

The ease of dynamic composition and interaction of course, comes not for free. It results in an additional memory and performance consumption in relation to a statically configured, simple message passing system which node IDs and event tags are specified on compile time. This overhead occurs in the native code for a certain platform and also in the additional run-time overhead of the MATLAB/Simulink code.

The current implementation of COSMIC for the AVR with CAN interface offers a CAN configuration protocol for the automatic assignment of node IDs, a binding protocol for mapping of the 64bit UID to a short network-specific name (event tag), a fragmentation protocol for the transmission of long messages as well as local event propagations. The functionality of this proof-of-concept has an overhead of 7kByte Flash and approximately 200 Bytes RAM memory in comparison with a statically organised system. The configuration and binding are executed in an initialisation phase outside of the critical communication path, so the time requirements of the actual event propagation correspond to a static implementation.

Considering the overhead for using COSMIC communication model in a distributed Matlab/Simulink implementation, the code size and the memory requirements are not a problem due to the performance of the PC platform. Moreover, the subscription runs outside the main control cycle as mentioned before and thus does not result in a run-time overhead. But also event handling has a very small overhead compared to raw TCP/IP communication. Incoming TCP/IP messages carrying events are handled by a call back function which reads them from the interface buffer, filters the ID for the subscribed channels and puts them into an associated buffer. Simulink fetches the most recent messages from there. This holds for any message whether it is a raw TCP/IP message or a COSMIC event. Hence, there is no temporal overhead when comparing a statically and a dynamically configured system. Reading, filtering and writing to a buffer requires around 0.02 ms for a message with a 64 Bit UID, 4 Byte data length value and 8 Bytes of payload. Of course, the stopwatch timer functions of Matlab/Simulink are non deterministic and depends on the PC and other running tasks. Hence, generally valid statements are here not possible.

4. EXPLOITING THE SCHEME IN THE DESIGN OF A SERVICE ROBOT

Our approach is used in the development process for a complex service robot control software. We are aiming at deriving a stand-alone version for the application by a stepwise migration of simulated sentient objects to native code. As described in the introduction, we are developing a versatile service robot. At the moment, this robot exists as a simulated version and as a mechanical prototype equipped with a distributed network of tiny micro-controllers performing dedicated control functions. Figure 6 shows the platform and also the distributed control hardware using the same basic components as the small experimental setup described above. Additionally, we provide communication via a standard wireless TCP/IP connection. The robot has four independent drives that can speed and turn each wheel independently. This enables very complex movement patterns generated by sophisticated control algorithms. It is by far more convenient and cost efficient to develop these mobility patterns in virtual reality than directly on a physical robot. Therefore we established a connection to enable interoperation between the simulation and the real hardware/mechanical components. We thus are able to use simulated and real components (in fact the robot can suffer from severe damage if wrong control signals are applied). Only when a simulation validates the control algorithm, it is applied to the real component. Each motor controller is modelled as sentient object. They subscribe to the motor command events disseminated by the respective control instance. Additonally, they publish events representing their local encoder data. Distance sensors for obstacle recognition and avoidance disseminate their information as shown in Figure 6.

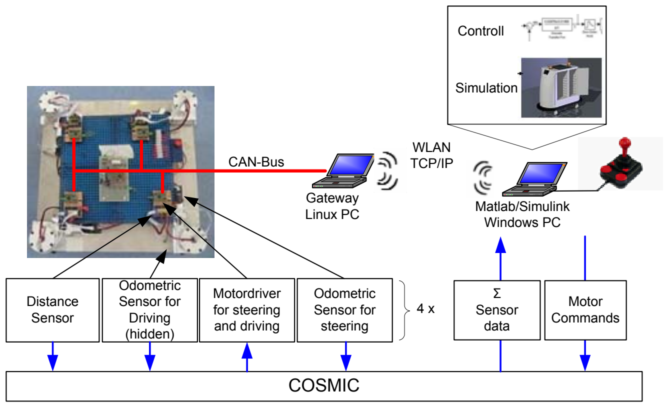

Fig. 6: Interaction between components

Such flexible combination of real sensors with Matlab\Simulink provides the user a tool chain supporting each development phase. The first step in the development cycle for such a control system is the data acquisition of the step response for each driving unit. Hence, respective ramp motor commands are published while the encoder signals are recorded. Based on these values an adequate plant model was developed with the control engineering tools of Matlab/Simulink. Afterwards the model was tested in Simulink for control parameter adaptation. These controllers for velocity and position were connected to the real robot in a next development step. This corresponds to a Software in the Loop scenario facilitating the fine tuning of the controller. The controller runs in Matlab with a period of 20 ms resulting from the achievable control cycle. To get to a lower control period the controllers were transformed into C code and installed on the microcontroller of each driving unit. Now the Matlab application was used for monitoring and calculation of references for the robot. For the development of semi-automated movements the robot were simulated in 3-D virtual reality tool. The user steer the robot system by a joystick. Hence the user controls speed and direction of the robot in the simulation, for the real robot or for both. The simulation is used to adapt the joystick behaviour to the user expectations without any danger for the real system.

In all development steps mentioned before the COSMIC middleware provides a general and common interface, which allows a flexible structure for investigations. To achieve our goals concerning the modular composition of reusable components, the monolithic block of movement calculation will be sub-structured into more fine grained components which can be encapsulated into sentient objects as presented in the small example above.

5. CONCLUSION

The paper presented an approach to support the development of complex control applications. As an example, we described a service robot used in the more and more important area of health care to deliver meals in a hospital. We brought together two well known approaches to raise the level of abstraction when modelling the complex robot behaviour and derive the control programs. On the one side there is a standard tool like Matlab/Simulink to define functional blocks and link these blocks to specify the overall robot behaviour. This is supported by a graphical user interface and a rich library of functions eases the development of complex computational tasks like filters and evaluators. On the other side we exploit the modularity and adaptability of a distributed hardware system composed from smart components like sensors, actuators and computational devices. With the availability of cheap computing and communication facilities like microcontrollers with integrated network access, such a system architecture is not only feasible but also has advantages in terms of extensibility and maintainability. We introduced the notion of a sentient object to model and event-based communication as an appropriate programming model for such system reflecting the modularity and composability of the hardware devices. It is shown in the paper how we integrated this with the standard

Matlab/Simulink development system. As a result we observe two major advantages. Firstly, we can identify reusable components with a well defined interface that can easily be combined because of the event-based interaction model. Secondly, we achieved interoperability between simulated components and real hardware. This substantially supports the development of complex control systems.

6. ACKNOWLEDGEMENTS

This work is part of the project DECOMOR which is sponsored by the DAAD and GRISCES in a German-Portuguese collaboration scheme.

REFERENCES

Bacon, J., Moody, K. Bates, J. Hayton, R. Ma, C. McNeil, A. Seidel, O. and Spiteri. M. (2000) Generic support for distributed applications. *IEEE Computer*, 33(3):68-76

Biegel, G. (2006), A Programming Model for Mobile, Context-Aware Applications, *PhD Thesis, Trinity College Dublin,* available as: TCD-CS-2006-59.pdf, from http://www.tara.tcd.ie/handle/2262/3298

Carreira, F., Canas, T., Silva A., Cardeira, C., (2006) "I-MERC: A mobile robot to deliver meals inside health services", in *Proceedings of RAM 2006, the IEEE International Conference on Robotics, Automation and Mechatronics,* 7 - 9 JUNE 2006, Bangkok, Thailand.

Casimiro, A., Kaiser,J., Verissimo P.,(2004), An Architectural Framework and a Middleware for Cooperating Smart Components, *ACM Computing Frontiers conference, CF '04*, ISCIA, Italy, 14-16 April 2004

Evans, J.; Krishnamurthy, B.; Barrows, B.; Skewis, T.; Lumelsky, V. (1992)., Handling real-world motion planning: a hospital transport robot, *Control Systems Magazine, IEEE*, vol. 12, Issue 1, pp 15 – 19

Fitzpatrick, A., Biegel, G., Clarke, S., Cahill, V. (2002), Towards a Sentient Object model. in *Workshop on Engineering Context-Aware Object-Oriented Systems and Environments (ECOOSE)*, Seattle, WA, USA, November, 2002.

Hopper, A., (2000), The Clifford Paterson Lecture, 1999 "Sentient computing", *Philosophical Transactions of the Royal Society London*, *358(1773):2349-2358*, Aug. 2000.

Kaiser, J., Mock, M., (1999) Implementing the real-timepublisher/subscriber model on the controller areanetwork (CAN). In *Proceedings of the 2ndInternational Symposium on Object-oriented Real-timedistributed Computing (ISORC99)*, Saint-Malo, France.

Kaiser, J., Pereira, C.E., Becker L.B., Villela C., Mitidieri C., (2001), On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects, *Proc. of the IEEE 4th International Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001),* Magdeburg, Germany, May 2001

Kaiser, J., Brudna, C., Mitidieri, C., Pereira C.E., (2003), COSMIC: A middleware for event-based interaction on CAN, *Proc. 9th IEEE Intern. Conference on Emerging Technologies and Factory Automation (ETFA2003),* Lisbon, Portugal.

Keutzer K., Malik S., Newton A. R., Rabaey J. M., and Sangiovanni-Vincentelli A., *System Level Design: Orthogonalization of Concerns and Platform-Based Design*, invited paper, IEEE Transactions on Computer-Aided Design, Vol. 19, No. 12, December 2000.

Swisslog (2004). Automatic guide vehicles provide bulk material transport in hospitals. http://www.swisslog.com/hcs-index/hcs-systems/hcs-agv.htm. Swisslog, Denver

Szyperski C. (1998) *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley

Verissimo P., Cahill, V., Casimiro, A., Cheverst K., Friday A., Kaiser, J., (2002), CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities, in *Proceedings of the European Wireless Conference*. Florence, Italy