An Architectural Framework and a Middleware for Cooperating Smart Components [*]

António Casimiro
U.Lisboa
casim@di.fc.ul.pt

Jörg Kaiser
U.Ulm
kaiser@informatik.uni-ulm.de

Paulo Veríssimo
U.Lisboa
pjv@di.fc.ul.pt

ABSTRACT

In a future networked physical world, a myriad of smart sensors and actuators assess and control aspects of their environments and autonomously act in response to it. Examples range in telematics, traffic management, team robotics or home automation to name a few. To a large extent, such systems operate proactively and independently of direct human control driven by the perception of the environment and the ability to organize respective computations dynamically. The challenging characteristics of these applications include sentience and autonomy of components, issues of responsiveness and safety criticality, geographical dispersion, mobility and evolution. A crucial design decision is the choice of the appropriate abstractions and interaction mechanisms. Looking to the basic building blocks of such systems we may find components which comprise mechanical components, hardware and software and a network interface, thus these components have different characteristics compared to pure software components. They are able to spontaneously disseminate information in response to events observed in the physical environment or to events received from other component via the network interface. Larger autonomous components may be composed recursively from these building blocks.

The paper describes an architectural framework and a middleware supporting a component-based system and an integrated view on events-based communication comprising the real world events and the events generated in the system. It starts by an outline of the component-based system construction. The generic event architecture GEAR is introduced which describes the event-based interaction between the components via a generic event layer. The generic event layer hides the different communication channels including the interactions through the environment. An appropriate middleware is presented which reflects these needs and allows to specify events which have quality attributes to express temporal constraints. This is complemented by the notion of event channels which are abstractions of the underlying network and allow to enforce quality attributes. They are established prior to interaction to reserve the needed computational and network resources for highly predictable event dissemination.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; C.2.4 []: Distributed Systems—*Distributed applications*; C.3 [**Special-Purpose and Application-Based Systems**]: *Real-Time and embedded systems*

General Terms

Design

Keywords

Events-based communication, sentient computing, component-based systems, middleware architectures

1. INTRODUCTION

In recent years we have seen the continuous improvement of technologies that are relevant for the construction of distributed embedded systems, including trustworthy visual, auditory, and location sensing [11], communication and processing. We believe that in a future networked physical world a new class of applications will emerge, composed of a myriad of smart sensors and actuators to assess and control aspects of their environments and autonomously act in response to it. The anticipated challenging characteristics of these applications include autonomy, responsiveness and safety criticality, large scale, geographical dispersion, mobility and evolution.

In order to deal with these challenges, it is of fundamental importance to use adequate high-level models, abstractions and interaction paradigms. Unfortunately, when facing the specific characteristics of the target systems, the shortcomings of current architectures and middleware interaction paradigms become apparent. Looking to the basic building blocks of such systems we may find components which comprise mechanical parts, hardware, software and a network interface. However, classical event/object models are usually software oriented and, as such, when trans-

[*]This work was partially supported by the EC, through project IST-2000-26031 (CORTEX), and by the FCT, through the Large-Scale Informatic Systems Laboratory (LaSIGE) and project POSI/1999/CHS/33996 (DEFEATS).

ported to a real-time, embedded systems setting, their harmony is cluttered by the conflict between, on the one side, send/receive of "software" events (message-based), and on the other side, input/output of "hardware" or "real-world" events, register-based. In terms of interaction paradigms, and although the use of event-based models appears to be a convenient solution [10, 22], these often lack the appropriate support for non-functional requirements like reliability, timeliness or security.

This paper describes an architectural framework and a middleware, supporting a component-based system and an integrated view on event-based communication comprising the real world events and the events generated in the system.

When choosing the appropriate interaction paradigm it is of fundamental importance to address the challenging issues of the envisaged sentient applications. Unlike classical approaches that confine the possible interactions to the application boundaries, i.e. to its components, we consider that the environment surrounding the application also plays a relevant role in this respect. Therefore, the paper starts by clarifying several issues concerning our view of the system, about the interactions that may take place and about the information flows. This view is complemented by providing an outline of the component-based system construction and, in particular, by showing that it is possible to compose larger applications from basic components, following an hierarchical composition approach.

This provides the necessary background to introduce the **Generic-Events Architecture (GEAR)**, which describes the event-based interaction between the components via a generic event layer while allowing the seamless integration of physical and computer information flows. In fact, the generic event layer hides the different communication channels, including the interactions through the environment.

Additionally, the event layer abstraction is also adequate for the proper handling of the non-functional requirements, namely reliability and timeliness, which are particularly stringent in real-time settings. The paper devotes particular attention to this issue by discussing the temporal aspects of interactions and the needs for predictability.

An appropriate middleware is presented which reflects these needs and allows to specify events which have quality attributes to express temporal constraints. This is complemented by the notion of *Event Channels (EC)*, which are abstractions of the underlying network while being abstracted by the event layer. In fact, event channels play a fundamental role in securing the functional and non-functional (e.g. reliability and timeliness) properties of the envisaged applications, that is, in allowing the enforcement of quality attributes. They are established prior to interaction to reserve the needed computational and network resources for highly predictable event dissemination.

The paper is organized as follows. In Section 3 we introduce the fundamental notions and abstractions that we adopt in this work to describe the interactions taking place in the system. Then, in Section 4, we describe the component-based approach that allows composition of objects. GEAR is then described in Section 5 and Section 6 focuses on temporal aspects of the interactions. Section 7 describes the COSMIC middleware, which may be used to specify the interaction between sentient objects. A simple example to highlight the ideas presented in the paper appears in Section 8 and Section 9 concludes the paper.

2. RELATED WORK

Our work considers a wired physical world in which a very large number of autonomous components cooperate. It is inspired by many research efforts in very different areas. Event-based systems in general have been introduced to meet the requirements of applications in which entities spontaneously generate information and disseminate it [1, 25, 22]. Intended for large systems and requiring quite complex infrastructures, these event systems do not consider stringent quality aspects like timeliness and dependability issues. Secondly, they are not created to support inter-operability between tiny smart devices with substantial resource constraints.

In [10] a real-time event system for CORBA has been introduced. The events are routed via a central event server which provides scheduling functions to support the real-time requirements. Such a central component is not available in an infrastructure envisaged in our system architecture and the developed middleware TAO (The Ace Orb) is quite complex and unsuitable to be directly integrated in smart devices.

There are efforts to implement CORBA for control networks, tailored to connect sensor and actuator components [15, 19]. They are targeted for the CAN-Bus [9], a popular network developed for the automotive industry. However, in these approaches the support for timeliness or dependability issues does not exist or is only very limited.

A new scheme to integrate smart devices in a CORBA environment is proposed in [17] and has lead to the proposal of a standard by the Object Management Group (OMG) [26]. Smart transducers are organized in clusters that are connected to a CORBA system by a gateway.

The clusters form isolated subnetworks. A special master node enforces the temporal properties in the cluster subnet. A CORBA gateway allows to access sensor data and write actuator data by means of an interface file system (IFS). The basic structure is similar to the WAN-of-CANs structure which has been introduced in the CORTEX project [4]. Islands of tight control may be realized by a control network and cooperate via wired or wireless networks covering a large number of these subnetworks. However, in contrast to the event channel model introduced in this paper, all communication inside a cluster relies on a single technical solution of a synchronous communication channel. Secondly, although the temporal behaviour of a single cluster is rigorously defined, no model to specify temporal properties for cluster-to-CORBA or cluster-to-cluster interactions is provided.

3. INFORMATION FLOW AND INTERACTION MODEL

In this paper we consider a component-based system model that incorporates previous work developed in the context of the IST CORTEX project [5]. As mentioned above, a fundamental idea underlying the approach is that applications can be composed of a large number of smart components that are able to sense their surrounding environment and interact with it. These components are referred to as *sentient objects*, a metaphor elaborated in CORTEX and inspired on the generic concept of *sentient computing* introduced in [12]. Sentient objects accept input events from a variety of different sources (including sensors, but not constrained to that), process them, and produce output events, whereby they actuate on the environment and/or interact with other objects. Therefore, the following kinds of interactions can take place in the system:

Environment-to-object interactions: correspond to a flow of information from the environment to application objects, reporting about the state of the former, and/or notifying about events taking place therein.

Object-to-object interactions: correspond to a flow of information among sentient objects, serving two purposes. The first is related with complementing the assessment of each individual object about the state of the surrounding space. The second is related to collaboration, in which the object tries to influence other objects into contributing to a common goal, or into reacting to an unexpected situation.

Object-to-environment interactions: correspond to a flow of information from an object to the environment, with the purpose of forcing a change in the state of the latter.

Before continuing, we need to clarify a few issues with respect to these possible forms of interaction. We consider that the environment can be a producer or consumer of information while interacting with sentient objects. The environment is the real (physical) world surrounding an object, not necessarily close to the object or limited to certain boundaries. Quite clearly, the information produced by the environment corresponds to the physical representation of real-time entities, of which typical examples include temperature, distance or the state of a door. On the other hand, actuation on the environment implies the manipulation of these real-time entities, like increasing the temperature (applying more heat), changing the distance (applying some movement) or changing the state of the door (closing or opening it). The required transformations between system representations of these real-time entities and their physical representations is accomplished, generically, by sensors and actuators. We further consider that there may exist *dumb* sensors and actuators, which interact with the objects by disseminating or capturing raw transducer information, and *smart* sensors and actuators, with enhanced processing capabilities, capable of "speaking" some more elaborate "event dialect" (see Sections 5 and 6.1). Interaction with the environment is therefore done through sensors and actuators, which may, or may not be part of sentient objects, as discussed in Section 4.2.

State or state changes in the environment are considered as events, captured by sensors (in the environment or within sentient objects) and further disseminated to other potentially interested sentient objects in the system. In consequence, it is quite natural to base the communication and interaction among sentient objects and with the environment on an event-based communication model. Moreover, typical properties of event-based models, such as anonymous and non-blocking communication, are highly desirable in systems where sentient objects can be mobile and where interactions are naturally very dynamic.

A distinguishing aspect of our work from many of the existing approaches, is that we consider that sentient objects may indirectly communicate with each other through the environment, when they act on it. Thus the environment constitutes an interaction and communication channel and is in the control and awareness loop of the objects. In other words, when a sentient object actuates on the environment it will be able to observe the state changes in the environment by means of events captured by the sensors. Clearly, other objects might as well capture the same events, thus establishing the above-mentioned indirect communication path.

In systems that involve interactions with the environment it is very important to consider the possibility of communication through the environment. It has been shown that the hidden channels developing through the latter (e.g., feedback loops) may hinder software-based algorithms ignoring them [30]. Therefore, any solution to the problem requires the definition of convenient abstractions and appropriate architectural constructs.

On the other hand, in order to deal with the information flow through the whole computer system and environment in a seamless way, handling "software" and "hardware" events uniformly, it is also necessary to find adequate abstractions. As discussed in Section 5, the Generic-Events Architecture introduces the concept of *Generic Event* and an *Event Layer* abstraction which aim at dealing, among others, with these issues.

4. SENTIENT OBJECT COMPOSITION

In this section we analyze the most relevant issues related with the sentient object paradigm and the construction of systems composed of sentient objects.

4.1 Component-based System Construction

Sentient objects can take several different forms: they can simply be software-based components, but they can also comprise mechanical and/or hardware parts, amongst which the very sensorial apparatus that substantiates "sentience", mixed with software components to accomplish their task. We refine this notion by considering a sentient object as an encapsulating entity, a component with internal logic and active processing elements, able to receive, transform and produce new events. This interface hides the internal hardware/software structure of the object, which may be complex, and shields the system from the low-level functional and temporal details of controlling a specific sensor or actuator.

Furthermore, given the inherent complexity of the envisaged applications, the number of simultaneous input events and the internal size of sentient objects may become too large and difficult to handle. Therefore, it should be possible to consider the hierarchical composition of sentient objects so that the application logic can be separated across as few or as many of these objects as necessary. On the other hand, composition of sentient objects should normally be constrained by the actual hardware component's structure, preventing the possibility of arbitrarily composing sentient objects. This is illustrated in Figure 1, where a sentient object is internally composed of a few other sentient objects, each of them consuming and producing events, some of which only internally propagated.

Observing the figure, and recalling our previous discussion about the possible interactions, we identify all of them here: an object-to-environment interaction occurs between the object controlling a WLAN transmitter and some WLAN receiver in the environment; an environment-to-object interaction takes place when the object responsible for the GPS

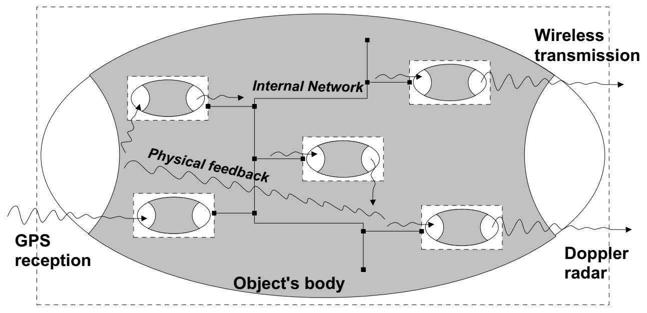


Figure 1: Component-aware sentient object composition.

signal reception uses the information transmitted by the satellites; finally, explicit object-to-object interactions occur internally to the container object, through an internal communication network. Additionally, it is interesting to observe that implicit communication can also occur, whether the physical feedback develops through the environment internal to the container object (as depicted) or through the environment external to this object. However, there is a subtle difference between both cases. While in the former the feedback can only be perceived by objects internal to the container, bounding the extent to which consistency must be ensured, such bounds do not exist in the latter. In fact, the notion of sentient object as an encapsulating entity may serve other purposes (e.g., the confinement of feedback and of the propagation of events), beyond the mere hierarchical composition of objects.

To give a more concrete example of such *component-aware* object composition we consider a scenario of cooperating robots. Each robot is made of several components, corresponding, for instance, to axis and manipulator controllers. Together with the control software, each of these controllers may be a sentient object. On the other hand, a robot itself is a sentient object, composed of the objects materialized by the controllers, and the environment internal to its own structure, or *body*.

This means that it should be possible to define cooperation activities using the events produced by robot sentient objects, without the need to know the internal structure of robots, or the events produced by body objects or by smart sensors within the body. From an engineering point of view, however, this also means that robot sentient object may have to generate new events that reflect its internal state, which requires the definition of a gateway to make the bridge between the internal and external environments.

4.2 Encapsulation and Scoping

Now an important question is about how to represent and disseminate events in a large scale networked world. As we have seen above, any event generated by a sentient object could, in principle, be visible anywhere in the system and thus received by any other sentient object. However, there are substantial obstacles to such universal interactions, originating from the components heterogeneity in such a large-scale setting.

Firstly, the components may have severe performance constraints, particularly because we want to integrate smart sensors and actuators in such an architecture. Secondly, the bandwidth of the participating networks may vary largely. Such networks may be low power, low bandwidth fieldbuses, or more powerful wireless networks as well as high speed backbones. Thirdly, the networks may have widely different reliability and timeliness characteristics. Consider a platoon of cooperating vehicles. Inside a vehicle there may be a field-bus like CAN [8, 9], TTP/A [17] or LIN [20], with a comparatively low bandwidth. On the other hand, the vehicles are communicating with others in the platoon via a direct wireless link. Finally, there may be multiple platoons of vehicles which are coordinated by an additional wireless network layer.

At the abstraction level of sentient objects, such heterogeneity is reflected by the notion of *body-vs-environment*. At the network level, we assume the *WAN-of-CANs* structure [27] to model the different networks. The notion of body and environment is derived from the recursively defined component-based object model. A body is similar to a cell membrane and represents a quality of service container for the sentient objects inside. On the network level, it may be associated with the components coupled by a certain CAN. A CAN defines the dissemination quality which can be expected by the cooperating objects.

In the above example, a vehicle may be a sentient object, whose body is composed of the respective lower level objects (sensors and actuators) which are connected by the internal network (see Figure 1). Correspondingly, the platoon can be seen itself as an object composed of a collection of cooperating vehicles, its body being the environment encapsulated by the platoon zone. At the network level, the wireless network represents the respective CAN. However, several platoons united by their CANs may interact with each other and objects further away, through some wider-range, possible fixed networking substrate, hence the concept of WAN-of-CANs.

The notions of body-environment and WAN-of-CANs are very useful when defining interaction properties across such boundaries. Their introduction obeyed to our belief that a single mechanism to provide quality measures for interactions is not appropriate. Instead, a high level construct for interaction across boundaries is needed which allows to specify the quality of dissemination and exploits the knowledge about body and environment to assess the feasibility of quality constraints. As we will see in the following section, the notion of an *event channel* represents this construct in our architecture. It disseminates events and allows the network independent specification of quality attributes. These attributes must be mapped to the respective properties of the underlying network structure.

5. A GENERIC EVENTS ARCHITECTURE

In order to successfully apply event-based object-oriented models, addressing the challenges enumerated in the introduction of this paper, it is necessary to use adequate architectural constructs, which allow the enforcement of fundamental properties such as timeliness or reliability.

We propose the **Generic-Events Architecture (GEAR)**, depicted in Figure 2, which we briefly describe in what follows (for a more detailed description please refer to [29]). The L-shaped structure is crucial to ensure some of the properties described.

Environment: The physical surroundings, remote and close, solid and etherial, of sentient objects.

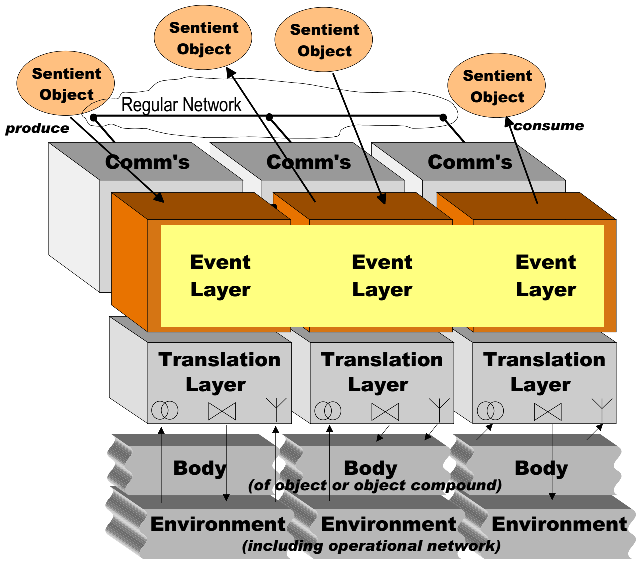


Figure 2: Generic-Events architecture.

Body: The physical embodiment of a sentient object (e.g., the hardware where a mechatronic controller resides, the physical structure of a car). Note that due to the compositional approach taken in our model, part of what is "environment" to a smaller object seen individually, becomes "body" for a larger, containing object. In fact, the body is the "internal environment" of the object. This architecture layering allows composition to take place seamlessly, in what concerns information flow.

Inside a body there may also be implicit knowledge, which can be exploited to make interaction more efficient, like the knowledge about the number of cooperating entities, the existence of a specific communication network or the simple fact that all components are co-located and thus the respective events do not need to specify location in their context attributes. Such intrinsic information is not available outside a body and, therefore, more explicit information has to be carried by an event.

Translation Layer: The layer responsible for physical event transformation from/to their native form to event channel dialect, between environment/body and an event channel. Essentially one doing observation and actuation operations on the lower side, and doing transactions of event descriptions on the other. On the lower side this layer may also interact with dumb sensors or actuators, therefore "talking" the language of the specific device. These interactions are done through *operational networks* (hence the antenna symbol in the figure).

Event Layer: The layer responsible for event propagation in the whole system, through several *Event Channels (EC)*:. In concrete terms, this layer is a kind of middleware that provides important event-processing services which are crucial for any realistic event-based system. For example, some of the services that imply the processing of events may include publishing, subscribing, discrimination (zoning, filtering, fusion, tracing), and queuing.

Communication Layer: The layer responsible for "wrapping" events (as a matter of fact, event descriptions in EC dialect) into "carrier" *event-messages*, to be transported to remote places. For example, a sensing event generated by a smart sensor is wrapped in an event-message and disseminated, to be caught by whoever is concerned. The same holds for an actuation event produced by a sentient object, to be delivered to a remote smart actuator. Likewise, this may apply to an event-message from one sentient object to another. Dumb sensors and actuators do not send event-messages, since they are unable to understand the EC dialect (they do not have an event layer neither a communication layer— they communicate, if needed, through operational networks).

Regular Network: This is represented in the horizontal axis of the block diagram by the communication layer, which encompasses the usual LAN, TCP/IP, and real-time protocols, desirably augmented with reliable and/or ordered broadcast and other protocols.

The GEAR introduces some innovative ideas in distributed systems architecture. While serving an object model based on production and consumption of generic events, it treats events produced by several sources (environment, body, objects) in a homogeneous way. This is possible due to the use of a common basic dialect for talking about events and due to the existence of the translation layer, which performs the necessary translation between the physical representation of a real-time entity and the EC compliant format. Crucial to the architecture is the event layer, which uses event channels to propagate events through regular network infrastructures. The event layer is realized by the COSMIC middleware, as described in Section 7.

5.1 Information Flow in GEAR

The flow of information (external environment and computational part) is seamlessly supported by the L-shaped architecture. It occurs in a number of different ways, which demonstrates the expressiveness of the model with regard to the necessary forms of information encountered in real-time cooperative and embedded systems.

Smart sensors produce events which report on the environment. Body sensors produce events which report on the body. They are disseminated by the local event layer module, on an event channel (EC) propagated through the regular network, to any relevant remote event layer modules where entities showed an interest on them, normally, sentient objects attached to the respective local event layer modules.

Sentient objects consume events they are interested in, process them, and produce other events. Some of these events are destined to other sentient objects. They are published on an EC using the same EC dialect that serves, e.g., sensor originated events. However, these events are semantically of a kind such that they are to be subscribed by the relevant sentient objects, for example, the sentient objects composing a robot controller system, or, at a higher level, the sentient objects composing the actual robots in a cooperative application. Smart actuators, on the other hand, merely consume events produced by sentient objects, whereby they accept and execute actuation commands. Alternatively to “talking” to other sentient objects, sentient objects can produce events of a lower level, for example, actuation commands on the body or environment. They publish these exactly the same way: on an event channel through the local event layer representative. Now, if these commands are of concern to local actuator units (e.g., body, including internal operational networks), they are passed on to the local translation layer. If they are of concern to a remote smart actuator, they are disseminated through the distributed event layer, to reach the former. In any case, if they are also of interest to other entities, such as other sentient objects that wish to be informed of the actuation command, then they are also disseminated through the EC to these sentient objects.

A key advantage of this architecture is that event-messages and physical events can be globally ordered, if necessary, since they all pass through the event layer. The model also offers opportunities to solve a long lasting problem in real-time, computer control, and embedded systems: the inconsistency between message passing and the feedback loop information flow subsystems.

6. TEMPORAL ASPECTS OF THE INTERACTIONS

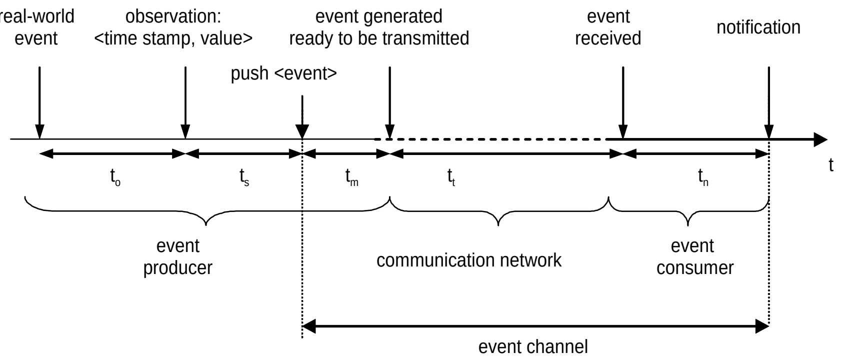
Any interaction needs some form of predictability. If safety critical scenarios are considered as it is done in CORTEX, temporal aspects become crucial and have to be made explicit. The problem is how to define temporal constraints and how to enforce them by appropriate resource usage in a dynamic ad-hoc environment. In an system where interactions are spontaneous, it may be also necessary to determine temporal properties dynamically. To do this, the respective temporal information must be stated explicitly and available during run-time. Secondly, it is not always ensured that temporal properties can be fulfilled. In these cases, adaptations and timing failure notification must be provided [2, 28]. In most real-time systems, the notion of a deadline is the prevailing scheme to express and enforce timeliness. However, a deadline only weakly reflect the temporal characteristics of the information which is handled. Secondly, a deadline often includes implicit knowledge about the system and the relations between activities. In a rather well defined, closed environment, it is possible to make such implicit assumptions and map these to execution times and deadlines. E.g. the engineer knows how long a vehicle position can be used before the vehicle movement outdates this information. Thus he maps this dependency between speed and position on a deadline which then assures that the position error can be assumed to be bounded. In a open environment, this implicit mapping is not possible any more because, as an obvious reason, the relation between speed and position, and thus the error bound, cannot easily be reverse engineered from a deadline. Therefore, our event model includes explicit quality attributes which allow to specify the temporal attributes for every individual event. This is of course an overhead compared to the use of implicit knowledge, but in a dynamic environment such information is needed.

To illustrate the problem, consider the example of the position of a vehicle. A position is a typical example for $\langle$*time, value*$\rangle$ entity [30]. Thus, the position is useful if we can determine an error bound which is related to time, e.g. if we want a position error below 10 meters to establish a safety property between cooperating cars moving with 5 m/sec, the position has a validity time of 2 seconds. In a $\langle$*time, value*$\rangle$ entity entity we can trade time against the precision of the value. This is known as value over time and time over value [18]. Once having established the time-value relation and captured in event attributes, subscribers of this event can locally decide about the usefulness of an information. In the GEAR architecture temporal validity is used to reason about safety properties in a event-based system [29]. We will briefly review the respective notions and see how they are exploited in our COSMIC event middleware.

Consider the timeline of generating an event representing some real-time entity [18] from its occurrence to the notification of a certain sentient object (Figure 3). The real-time entity is captured at the sensor interface of the system and has to be transformed in a form which can be treated by a computer. During the time interval t_0 the sensor reads the real-time entity and a time stamp is associated with the respective value. The derived $\langle$*time, value*$\rangle$ entity represents an observation. It may be necessary to perform substantial local computations to derive application relevant information from the raw sensor data. However, it should be noted that the time stamp of the observation is associated with the capture time and thus independent from further signal processing and event generation. This close relationship between capture time and the associated value is supported by smart sensors described above.

The processed sensor information is assembled in an event data structure after t_s to be published to an event channel. As is described later, the event includes the time stamp of generation and the temporal validity as attributes.

The temporal validity is an application defined measure for the expiration of a $\langle$*time, value*$\rangle$. As we explained in the example of a position above, it may vary dependent on application parameters. Temporal validity is a more general concept than that of a deadline. It is independent of a certain technical implementation of a system. While deadlines may be used to schedule the respective steps in an event generation and dissemination, a temporal validity is an intrinsic property of a $\langle$*time, value*$\rangle$ entity carried in an event. A temporal validity allows to reason about the usefulness of information and is beneficial even in systems in which timely dissemination of events cannot be enforced because it enables timing failure detection at the event consumer. It is obvious that deadlines or periods can be derived from the temporal validity of an event. To set a deadline, knowledge of an implementation, worst case execution times or message dissemination latencies is necessary. Thus, in the timeline of Figure 3 every interval may have a deadline. Event dissemination through soft real-time channels in COSMIC exploits the temporal validity to define dissemination deadlines. Quality attributes can be defined, for instance, in terms of $\langle$*validity interval, omission degree*$\rangle$ pairs. These allow to characterize the usefulness of the event for a certain application, in a certain context. Because of that, quality attributes of an event clearly depend on higher level issues, such as the nature of the sentient object or of the smart sensor that produced the event. For instance, an event containing an indication of some vehicle speed must have different quality attributes depending on the kind of vehicle



t_o	: time to obtain an observation
t_s	: time to process sensor reading
t_m	: time to assemble an event message
t_t	: time to transfer the event on the regular network
t_n	: time for notification on the consumer site

Figure 3: Event processing and dissemination.

from which it originated, or depending on its current speed. The same happens with the position event of the car example above, whose validity depends on the current speed and on a predefined required precision. However, since quality attributes are strictly related with the semantics of the application or, at least, with some high level knowledge of the purpose of the system (from which the validity of the information can be derived), the definition of these quality attributes may be done by exploiting the information provided at the programming interface. Therefore, it is important to understand how the system programmer can specify non-functional requirements at the API, and how these requirements translate into quality attributes assigned to events. While temporal validity is identified as an intrinsic event property, which is exploited to decide on the usefulness of data at a certain point in time, it is still necessary to provide a communication facility which can disseminate the event before the validity is expired.

In a WAN-of-CANs network structure we have to cope with very different network characteristics and quality of service properties. Therefore, when crossing the network boundaries the quality of service guarantees available in a certain network will be lost and it will be very hard, costly and perhaps impossible to achieve these properties in the next larger area of the WAN-of CANs structure. CORTEX has a couple of abstractions to cope with this situation (network zones, body/environment) which have been discussed above. From the temporal point of view we need a high level abstraction like the temporal validity for the individual event now to express our quality requirements of the dissemination over the network. The $\langle$*bound, coverage*$\rangle$ pair, introduced in relation with the TCB [28] seems to be an appropriate approach. It considers the inherent uncertainty of networks and allows to trade the quality of dissemination against the resources which are needed. In relation with the event channel model discussed later, the $\langle$*bound, coverage*$\rangle$ pair allows to specify the quality properties of an event channel independently of specific technical issues. Given the typical environments in which sentient applications will operate, where it is difficult or even impossible to provide timeliness or reliability guarantees, we proposed an alternative way to handle non-functional application requirements, in relation with the TCB approach [28]. The proposed approach exploits intrinsic characteristics of applications, such as fail-safety, or time-elasticity, in order to secure QoS specifications of the form $\langle$*bound, coverage*$\rangle$. Instead of constructing systems that rely on guaranteed bounds, the idea is to use (possibly changing) bounds that are secured with a constant probability all over the execution. This obviously requires an application to be able to adapt to changing conditions (and/or changing bounds) or, if this is not possible, to be able to perform some safety procedures when the operational conditions degrade to an unbearable level. The bounds we mentioned above refer essentially to timeliness bounds associated to the execution of local or distributed activities, or combinations thereof. From these bounds it is then possible to derive the quality attributes, in particular validity intervals, that characterize the events published in the event channel.

6.1 The Role of Smart Sensors and Actuators

Smart devices encapsulate hardware, software and mechanical components and provide information and a set of well specified functions and which are closely related to the interaction with the environment. The built-in computational components and the network interface enable the implementation of a well-defined high level interface that does not just provide raw transducer data, but a processed, application-related set of events. Moreover, they exhibit an autonomous spontaneous behaviour. They differ from general purpose nodes because they are dedicated to a certain functionality which complies to their sensing and actuating capabilities while general purpose node may execute any program.

Concerning the sentient object model, smart sensors and actuators may be basic sentient objects themselves, consuming events from the real-world environment and producing the respective generic events for the system's event layer or, vice versa consuming a generic event and converting it to a real-world event by an actuation. Smart components therefore constitute the periphery, i.e. the real-world interface of a more complex sentient object. The model of sentient objects also constitutes the framework to built more complex "virtual" sensors by relating multiple (primary, i.e. sensors which directly sense a physical entity) sensors.

Smart components translate events of the environment to an appropriate form available at the event layer or, vice versa, transform a system event into an actuation. For smart components we can assume that:

- Smart components have dedicated resources to perform a specific function.
- These resources are not used for other purposes during normal real-time operation.
- No local temporal conflicts occur that will change the observable temporal behaviour.
- The functions of a component can usually only be changed during a configuration procedure which is not performed when the component is involved in critical operations.
- An observation of the environment as a ⟨time,value⟩ pair can be obtained with a bounded jitter in time.

Many predictability and scheduling problems arise from the fact, that very low level timing behaviours have to be handled on a single processor. Here, temporal encapsulation of activities is difficult because of the possible side effects when sharing a single processor resource. Consider the control of a simple IR-range detector which is used for obstacle avoidance. Dependent on its range and the speed of a vehicle, it has to be polled to prevent the vehicle from crashing into an obstacle. On a single central processor, this critical activity has to be coordinated with many similar, possibly less critical functions. It means that a very fine grained schedule has to be derived based purely on the artifacts of the low level device control. In a smart sensor component, all this low level timing behaviour can be optimized and encapsulated. Thus we can assume temporal encapsulation similar to information hiding in the functional domain. Of course, there is still the problem to guarantee that an event will be disseminated and recognized in due time by the respective system components, but this relates to application related events rather than the low artifacts of a device timing. The main responsibility to provide timeliness guarantees is shifted to the event layer where these events are disseminated. Smart sensors thus lead to network centric system model. The network constitute the shared resource which has to be scheduled in a predictable way. The COSMIC middleware introduced in the next section is an approach to provide predictable event dissemination for a network of smart sensors and actuators.

7. AN EVENT MODEL AND MIDDLEWARE FOR COOPERATING SMART DEVICES

An event model and a middleware suitable for smart components must support timely and reliable communication and also must be resource efficient. **COSMIC (COoperating Smart devices)** is aimed at supporting the interaction between those components according to the concepts introduced so far. Based on the model of a WAN-of-CANs, we assume that the components are connected to some form of CAN as a fieldbus or a special wireless sensor network which provides specific network properties. E.g. a fieldbus developed for control applications usually includes mechanisms for predictable communication while other networks only support a best effort dissemination. A gateway connects these CANs to the next level in the network hierarchy. The event system should allow the dynamic interaction over a hierarchy of such networks and comply with the overall CORTEX generic event model. Events are typed information carriers and are disseminated in a publisher/ subscriber style [24, 7], which is particularly suitable because it supports generative, anonymous communication [3] and does not create any artificial control dependencies between producers of information and the consumers. This decoupling in space (no references or names of senders or receivers are needed for communication) and the flow decoupling (no control transfer occurs with a data transfer) are well known [24, 7, 14] and crucial properties to maintain autonomy of components and dynamic interactions.

It is obvious that not all networks can provide the same QoS guarantees and secondly, applications may have widely differing requirements for event dissemination. Additionally, when striving for predictability, resources have to be reserved and data structures must be set up before communication takes place. Thus, these things can not predictably be made on the fly while disseminating an event. Therefore, we introduced the notion of an event channel to cope with differing properties and requirements and have an object to which we can assign resources and reservations. The concept of an event channel is not new [10, 25], however, it has not yet been used to reflect the properties of the underlying heterogeneous communication networks and mechanisms as described by the GEAR architecture. Rather, existing event middleware allows to specify the priorities or deadlines of events handled in an event server. Event channels allow to specify the communication properties on the level of the event system in a fine grained way. An event channel is defined by:

$$event\_channel := \langle subject, quality\_attributeList, \\ handlers \rangle$$

The subject determines the types of events event which may be issued to the channel. The quality attributes model the properties of the underlying communication network and dissemination scheme. These attributes include latency specifications, dissemination constraints and reliability parameters. The notion of zones which represent a guaranteed quality of service in a subnetwork support this approach. Our goal is to handle the temporal specifications as ⟨*bound, coverage*⟩ pairs [28] orthogonal to the more technical questions of how to achieve a certain synchrony property of the dissemination infrastructure. Currently, we support quality attributes of event channels in a CAN-Bus environment represented by explicit synchrony classes.

The COSMIC middleware maps the channel properties to lower level protocols of the regular network. Based on our previous work on predictable protocols for the CAN-Bus, COSMIC defines an abstract network which provides hard, soft and non real-time message classes [21].

Correspondingly, we distinguish three event channel classes according to their synchrony properties: hard real-time channels, soft real-time channels and non-real-time channels.

Hard real-time channels (HRTC) guarantee event propagation within the defined time constraints in the presence of a specified number of omission faults. HRTECs are supported by a reservation scheme which is similar to the scheme used in time-triggered protocols like TTP [16][31], TTP/A [17], and TTCAN [8]. However, a substantial advantage over a TDMA scheme is that due to CAN-Bus properties, bandwidth which was reserved but is not needed by a HRTEC can be used by less critical traffic [21].

Soft real-time channels (SRTC) exploit the temporal validity interval of events to derive deadlines for scheduling. The validity interval defines the point in time after which an event becomes temporally inconsistent. Therefore, in a real-time system an event is useless after this point and may me discarded. The transmission deadline (DL) is defined as the latest point in time when a message has to be transmitted and is specified in a time interval which is derived from the expiration time:

$$t_{event\_ready} < DL < t_{expiration} - \Delta_{notification}$$

$t_{expiration}$ defines the point in time when the temporal validity expires. $\Delta_{notification}$ is the expected end-to-end latency which includes the transfer time over the network and the time the event may be delayed by the local event handling in the nodes. As said before, event deadlines are used to schedule the dissemination by SRTECs. However, deadlines may be missed in transient overload situations or due to arbitrary arrival times of events. On the publisher side the application's exception handler is called whenever the event deadline expires before event transmission. At this point in time the event is also not expected to arrive at the subscriber side before the validity expires. Therefore, the event is removed from the sending queue. On the subscriber side the expiration time is used to schedule the delivery of the event. If the event cannot be delivered until its expiration time it is removed from the respective queues allocated by the COSMIC middleware. This prevents the communication system to be loaded by outdated messages.

Non-real-time channels do not assume any temporal specification and disseminate events in a best effort manner. An instance of an event channel is created locally, whenever a publisher makes an announcement for publication or a subscriber subscribes for an event notification. When a publisher announces publication, the respective data structures of an event channel are created by the middleware. When a subscriber subscribes to an event channel, it may specify context attributes of an event which are used to filter events locally. E.g. a subscriber may only be interested in events generated at a certain location. Additionally the subscriber specifies quality properties of the event channel. A more detailed description of the event channels can be found in [13].

Currently, COSMIC handles all event channels which disseminate events beyond the CAN network boundary as non real-time event channels. This is mainly because we use the TCP/IP protocol to disseminate events over wireless links or to the standard Ethernet. However, there are a number of possible improvements which can easily be integrated in the event channel model. The Timely Computing Base (TCB) [28] can be exploited for timing failure detection and thus would provide awareness for event dissemination in environments where timely delivery of events cannot be enforced. Additionally, there are wireless protocols which can provide timely and reliable message delivery [6, 23] which may be exploited for the respective event channel classes.

Events are the information carriers which are exchanged between sentient objects through event channels. To cope with the requirements of an ad-hoc environment, an event includes the description of the context in which it has been generated and quality attributes defining requirements for dissemination. This is particularly important in an open, dynamic environment where an event may travel over multiple networks. An event instance is specified as:

$$\begin{aligned} event := \langle & subject, context\_attributeList, \\ & quality\_attributeList, contents \rangle \end{aligned}$$

A subject defines the type of the event and is related to the event contents. It supports anonymous communication and is used to route an event. The subject has to match to the subject of the event channel through which the event is disseminated. Attributes are complementary to the event contents. They describe individual functional and non-functional properties of the event. The context attributes describe the environment in which the event has been generated, e.g. a location, an operational mode or a time of occurrence. The quality attributes specify timeliness and dependability aspects in terms of ⟨*validity interval, omission degree*⟩ pairs. The validity interval defines the point in time after which an event becomes temporally inconsistent [18]. As described above, the temporal validity can be mapped to a deadline. However, usually a deadline is an engineering artefact which is used for scheduling while the temporal validity is a general property of a ⟨*time, value*⟩ entity. In a environment where a deadline cannot be enforced, a consumer of an event eventually must decide whether the event still is temporally consistent, i.e. represents a valid ⟨*time, value*⟩ entity.

7.1 The Architecture of the COSMIC Middleware

On the architectural level, COSMIC distinguish three layers roughly depicted in Figure 4. Two of them, the event layer and the abstract network layer are implemented by the COSMIC middleware. The event layer provides the API for the application and realizes the abstraction of event and event channels.

The abstract network implements real-time message classes and adapts the quality requirements to the underlying real network. An event channel handler resides in every node. It supports the programming interface and provides the necessary data structures for event-based communication. Whenever an object subscribes to a channel or a publisher announces a channel, the event channel handler is involved. It initiates the binding of the channel's subject, which is represented by a network independent unique identifier to an address of the underlying abstract network to enable communication [14]. The event channel handler then tightly cooperates with the respective handlers of the abstract network layer to disseminate events or receive event notifications. It should be noted that the QoS properties of the event layer in general depend on what the abstract network layer can provide. Thus, it may not always be possible to e.g. support hard real-time event channels because the abstract network layer cannot provide the respective guarantees. In [13], we describe the protocols and services of the abstract network layer particularly for the CAN-Bus.

As can be seen in Figure 4, the hard real-time (HRT) message class is supported by a dedicated handler which is able to provide the time triggered message dissemination.

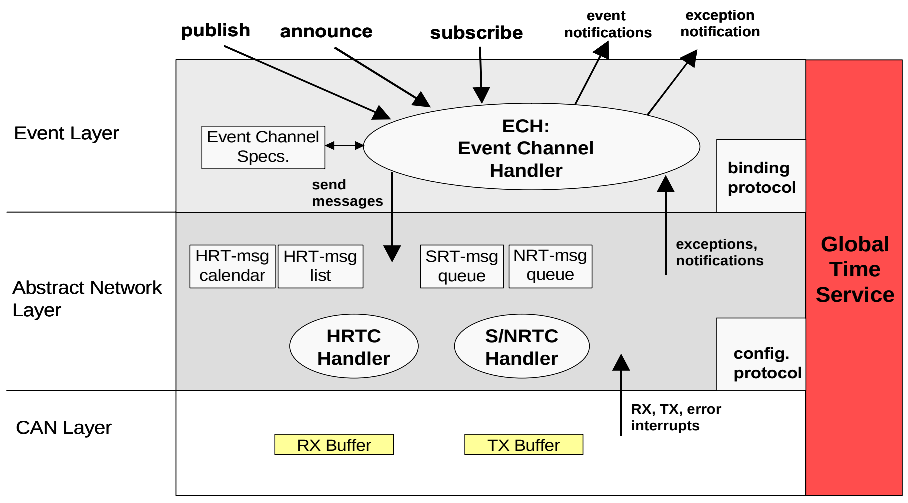


Figure 4: Architecture layers of COSMIC.

The HRT handler maintains the HRT message list, which contains an entry for each local HRT message to be sent. The entry holds the parameters for the message, the activation status and the binding information. Messages are scheduled on the bus according to the HRT message calendar which comprises the precise start time for each time slot allocated for a message. Soft real-time message queues order outgoing messages according to their transmission deadlines derived from the temporal validity interval. If the transmission deadline is exceeded, the event message is purged out of the queue. The respective application is notified via the exception notification interface and can take actions like trying to publish the event again or publish it to a channel of another class. Incoming event messages are ordered according to their temporal validity. If an event message arrive, the respective applications are notified. At the moment, an outdated message is deleted from the queue and if the queue runs out of space, the oldest message is discarded. However, there are other policies possible depending on event attributes and available memory space. Non real-time messages are FIFO ordered in a fixed size circular buffer.

7.2 Status of COSMIC

The goal for developing COSMIC was to provide a platform to seamlessly integrate smart tiny components in a large system. Therefore, COSMIC should run also on the small, resource constraint devices which are built around 16-Bit or even 8-Bit micro-controllers. The distributed COSMIC middleware has been implemented and tested on various platforms. Under RT-Linux, we support the real-time channels over the CAN Bus as described above. The RT-Linux version runs on Pentium processors and is currently evaluated before we intent to port it to a smart sensor or actuator. For the interoperability in a WAN-of-CANs environment, we only provide non real-time channels at the moment. This version includes a gateway between the CAN-bus and a TCP/IP network. It allows us to use a standard wireless 802.11 network. The non real-time version of COSMIC is available on Linux, RT-Linux and on the micro-controller families C167 (Infineon) and 68HC908 (Motorola). Both micro-controllers have an on-board CAN controller and thus do not require additional hardware components for the network. The memory footprint of COSMIC is about 13 Kbyte on a C167 and slightly more on the 68HC908 where it fits into the on-board flash memory without problems. Because only a few channels are required on such a smart sensor or actuator component, the requirement of RAM (which is a scarce resource on many single chip systems) to hold the dynamic data structures of a channel is low. The COSMIC middleware makes it very easy to include new smart sensors in an existing system. Particularly, the application running on a smart sensor to condition and process the raw physical data must not be aware of any low level network specific details. It seamlessly interacts with other components of the system exclusively via event channels.

The demo example, briefly described in the next chapter, is using a distributed infrastructure of tiny smart sensors and actuators directly cooperating via event channels over heterogeneous networks.

8. AN ILLUSTRATIVE EXAMPLE

A simple example for many important properties of the proposed system showing the coordination through the environment and events disseminated over the network is the demo of two cooperating robots depicted in Figure 5.

Each robot is equipped with smart distance sensors, speed sensors, acceleration sensors and one of the robots (the “guide” (KURT2) in front (Figure 5)) has a tracking camera allowing to follow a white line. The robots form a WAN-of-CANs system in which their local CANs are interconnected via a wireless 802.11 network. COSMIC provides the event layer for seamless interaction. The “blind” robot (N.N.) is searching the guide randomly. Whenever the blind robot detects (by its front distance sensors) an obstacle, it checks whether this may be the guide. For this purpose, it dynamically subscribes to the event channel disseminating distance events from rear distance sensors of the guide(s) and compares these with the distance events from its local front sensors. If the distance is approximately the same it infers that it is really behind a guide. Now N.N. also subscribes to the event channels of the tracking camera and the speed sensors

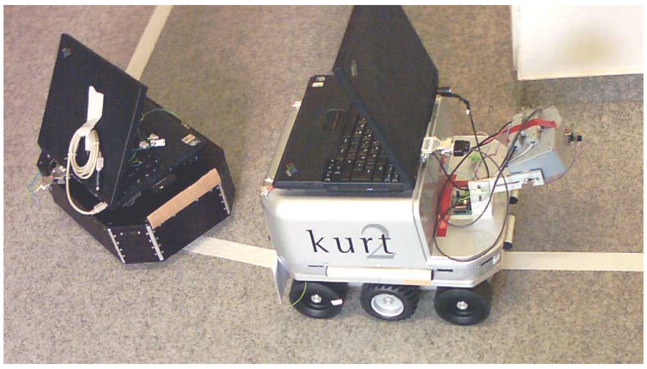


Figure 5: Cooperating robots.

to follow the guide. The demo application highlights the following properties of the system:

1. Dynamic interaction of robots which is not known in advance. In principle, any two a priori unknown robots can cooperate. All what publishers and subscribers have to know to dynamically interact in this environment is the subject of the respective event class. A problem will be to receive only the events of the robot which is closest. A robot identity does not help much to solve this problem. Rather, the position of the event generation entity which is captured in the respective attributes can be evaluated to filter the relevant event out of the event stream. A suitable wireless protocol which uses proximity to filter events has been proposed by Meier and Cahill [22] in the CORTEX project.

2. Interaction through the environment. The cooperation between the robots is controlled by sensing the distance between the robots. If the guide detects that the distance grows, it slows down. Respectively, if the blind robot comes too close it reduces its speed. The local distance sensors produce events which are disseminated through a low latency, highly predictable event channel. The respective reaction time can be calculated as function of the speed and the distance of the robots and define a dynamic dissemination deadline for events. Thus, the interaction through the environment will secure the safety properties of the application, i.e. the follower may not crash into the guide and the guide may not loose the follower. Additionally, the robots have remote subscriptions to the respective distance events which are used to check it with the local sensor readings to validate that they really follow the guide which they detect with their local sensors. Because there may be longer latencies and omissions, this check occasionally will not be possible. The unavailability of the remote events will decrease the quality of interaction and probably and slow down the robots, but will not affect safety properties.

3. Cooperative sensing. The blind robot subscribes to the events of the line tracking camera. Thus it can "see" through the eye of the guide. Because it knows the distance to the guide and the speed as well, it can foresee the necessary movements. The proposed system provides the architectural framework for such a cooperation. The respective sentient object controlling the actuation of the robot receives as input the position and orientation of the white line to be tracked. In the case of the guide robot, this information is directly delivered as a body event with a low latency and a high reliability over the internal network. For the follower robot, the information comes also via an event channel but with different quality attributes. These quality attributes are reflected in the event channel description. The sentient object controlling the actuation of the follower is aware of the increased latency and higher probability of omission.

9. CONCLUSION AND FUTURE WORK

The paper addresses problems of building large distributed systems interacting with the physical environment and being composed from a huge number of smart components. We cannot assume that the network architecture in such a system is homogeneous. Rather multiple "edge-" networks are fused to a hierarchical, heterogeneous wide area network. They connect the tiny sensors and actuators perceiving the environment and providing sentience to the application. Additionally, mobility and dynamic deployment of components require the dynamic interaction without fixed, a priori known addressing and routing schemes. The work presented in the paper is a contribution towards the seamless interaction in such an environment which should not be restricted by technical obstacles. Rather it should be possible to control the flow of information by explicitly specifying functional and temporal dissemination constraints.

The paper presented the general model of a sentient object to describe composition, encapsulation and interaction in such an environment and developed the Generic Event Architecture GEAR which integrates the interaction through the environment and the network. While appropriate abstractions and interaction models can hide the functional heterogeneity of the networks, it is impossible to hide the quality differences. Therefore, one of the main concerns is to define temporal properties in such an open infrastructure. The notion of an event channel has been introduced which allows to specify quality aspects explicitly. They can be verified at subscription and define a boundary for event dissemination. The COSMIC middleware is a first attempt to put these concepts into operation. COSMIC allows the interoperability of tiny components over multiple network boundaries and supports the definition of different real-time event channel classes.

There are many open questions that emerged from our work. One direction of future research will be the inclusion of real-world communication channels established between sensors and actuators in the temporal analysis and the ordering of such events in a cause-effect chain. Additionally, the provision of timing failure detection for the adaptation of interactions will be in the focus of our research. To reduce network traffic and only disseminate those events to the subscribers which they are really interested in and which have a chance to arrive timely, the encapsulation and scoping schemes have to be transformed into respective multi-level filtering rules. The event attributes which describe aspects of the context and temporal constraints for the dissemination will be exploited for this purpose. Finally, it is intended to integrate the results in the COSMIC middleware to enable experimental assessment.

10. REFERENCES

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

[2] L. B. Becker, M. Gergeleit, S. Schemmer, and E. Nett. Using a flexible real-time scheduling strategy in a distributed embedded application. In *Proc. of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Lisbon, Portugal, Sept. 2003.

[3] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, apr 1989.

[4] A. Casimiro (Ed.). Preliminary definition of cortex system architecture. CORTEX project, IST-2000-26031, Deliverable D4, Apr. 2002.

[5] CORTEX project Annex 1, Description of Work. Technical report, CORTEX project, IST-2000-26031, Oct. 2000. http://cortex.di.fc.ul.pt.

[6] R. Cunningham and V. Cahill. Time bounded medium access control for ad hoc networks. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02)*, pages 1–8, Toulouse, France, Oct. 2002. ACM Press.

[7] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical Report DSC ID:200104, EPFL, Lausanne, Switzerland, 2001.

[8] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M.Walther. Time triggered communication on CAN, 2000. http://www.can-cia.org/can/ttcan/fuehrer.pdf.

[9] R. B. GmbH. CAN Specification Version 2.0. Technical report, Sept. 1991.

[10] T. Harrison, D. Levine, and D. Schmidt. The design and performance of a real-time corba event service. In *Proceedings of the 1997 Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 184–200, Atlanta, Georgia, USA, 1997. ACM Press.

[11] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, aug 2001.

[12] A. Hopper. The Clifford Paterson Lecture, 1999 Sentient Computing. *Philosophical Transactions of the Royal Society London*, 358(1773):2349–2358, Aug. 2000.

[13] J. Kaiser, C. Mitidieri, C. Brudna, and C. Pereira. COSMIC: A Middleware for Event-Based Interaction on CAN. In *Proc. 2003 IEEE Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal, Sept. 2003.

[14] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the controller area network (CAN). In *Proceedings of the 2nd International Symposium on Object-oriented Real-time distributed Computing (ISORC99)*, Saint-Malo, France, May 1999.

[15] K. Kim, G. Jeon, S. Hong, T. Kim, and S. Kim. Integrating subscription-based and connection-oriented communications into the embedded CORBA for the CAN Bus. In *Proceedings of the IEEE Real-time Technology and Application Symposium*, May 2000.

[16] H. Kopetz and G. Grünsteidl. TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. Technical Report rr-12-92, Institut für Technische Informatik, Technische Universität Wien, Treilstr. 3/182/1, A-1040 Vienna, Austria, 1992.

[17] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. *International Journal of Computer System, Science Engineering*, 16(2), Mar. 2001.

[18] H. Kopetz and P. Veríssimo. Real-time and Dependability Concepts. In S. J. Mullender, editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 16, pages 411–446. Addison-Wesley, 1993.

[19] S. Lankes, A. Jabs, and T. Bemmerl. Integration of a CAN-based connection-oriented communication model into Real-Time CORBA. In *Workshop on Parallel and Distributed Real-Time Systems*, Nice, France, Apr. 2003.

[20] Local Interconnect Network: LIN Specification Package Revision 1.2. Technical report, Nov. 2000.

[21] M. Livani, J. Kaiser, and W. Jia. Scheduling hard and soft real-time communication in the controller area network. *Control Engineering*, 7(12):1515–1523, 1999.

[22] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*, pages 639–644, Vienna, Austria, 2002.

[23] E. Nett and S. Schemmer. Reliable real-time communication in cooperative mobile applications. *IEEE Transactions on Computers*, 52(2):166–180, Feb. 2003.

[24] B. Oki, M. Pfluegl, A. Seigel, and D. Skeen. The information bus - an architecture for extensible distributed systems. *Operating Systems Review*, 27(5):58–68, 1993.

[25] O. M. G. (OMG). CORBAservices: Common Object Services Specification - Notification Service Specification, Version 1.0, 2000.

[26] O. M. G. (OMG). Smart transducer interface, initial submission, June 2001.

[27] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser. Cortex: Towards supporting autonomous and cooperating sentient entities. In *Proceedings of European Wireless 2002*, Florence, Italy, Feb. 2002.

[28] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *Transactions on Computers - Special Issue on Asynchronous Real-Time Systems*, 51(8):916–930, Aug. 2002.

[29] P. Veríssimo and A. Casimiro. Event-driven support of real-time sentient objects. In *Proceedings of the 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, Guadalajara, Mexico, Jan. 2003.

[30] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.