Generic-Events Architecture: Integrating real-world aspects in event-based systems [⋆]

António Casimiro[1], Jörg Kaiser[2], and Paulo Verissimo[1]

[1] Univ. Lisboa, {casim,pjv}@di.fc.ul.pt
[2] Univ. Magdeburg, kaiser@ivs.cs.uni-magdeburg.de

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. To a large extent, such systems operate proactively and independently of direct human control. They include computer hardware and software parts mixed with mechanical devices. Besides the regular computer communication channels, they also establish interaction channels among them directly through the environment. These characteristics pose a number of fundamentally new consistency and correctness challenges which, if not met, may hinder the dependability of such systems, and ultimately lead to unexpected failures.
This paper describes an architectural framework and event model capable of solving these pressing problems. Firstly, we offer an innovative composable object model representing software/hardware entities capable of interacting with the environment. Secondly, we provide event-based communication seamlessly integrating real-world events and events generated in the system. The crucial parts of our work are the generic-events architecture GEAR, hosting the COSMIC middleware supporting the events model, with attributes to express spatial and temporal properties.

1 Introduction

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. In fact, this is also made possible by the continuous improvement of technologies that are relevant for the construction of these systems, including trustworthy visual, auditory, and location sensing [1], communication and processing. To a large extent, such systems operate proactively and independently of direct human control, instead driven by the perception of the environment and the ability to organize their own computations and actuations dynamically. The challenging characteristics of these applications include sentience and autonomy of components, issues of responsiveness and safety criticality, geographical dispersion, mobility and evolution. In order to deal with these

[⋆] This work was partially supported by the EC, through project IST-FP6-STREP-26979 (HIDENETS), and by the FCT, through the Large-Scale Informatic Systems Laboratory (LaSIGE).

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.

As basic building blocks of such systems we often find computer hardware and software parts mixed with mechanical devices, and one or more network interfaces. In consequence, these components have different characteristics compared to pure software components. Moreover, these artifacts, besides the regular computer communication channels, also establish interaction channels among them directly through the environment. They are able to spontaneously disseminate information in response to events observed in the physical environment or to events received from other components. Larger autonomous components may be composed recursively from these building blocks.

However, classical event/object models are usually software oriented and, as such, when transported 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 [2, 3], these often lack the appropriate support for non-functional requirements like reliability, timeliness or security.

The afore-mentioned characteristics pose a number of fundamentally new consistency and correctness challenges which, if not met, may hinder the dependability of such systems, and ultimately lead to unexpected failures. We believe that the first step in solving the former lies on an architecture allowing to: express and represent the software/hardware structure of these components and their composability; bridge the consistency gap between the events representing the physical environment generated by sensor readings and the events encapsulating state changes resulting from computations of the system.

This paper describes an architectural framework and event model capable of solving these pressing problems. Firstly, we offer an innovative composable object model which represents software/hardware entities capable of interacting with the environment. Secondly, we provide event-based communication seamlessly integrating real-world events and events generated in the system. After providing an overview of related work, 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. The crucial parts of our work, the generic-events architecture GEAR, hosting the COSMIC middleware supporting the events model, are then introduced.

The Generic-Events Architecture (GEAR) describes the event-based interaction between the components via a generic event layer. This layer integrates different communication channels including the interactions through the environment. Because interaction with the physical world requires real-time properties, concepts of time describing the aging of information and temporal consistency have to be included, in order to allow correct representation of these interactions by algorithms. The paper devotes particular attention to this issue by discussing the temporal aspects of interactions and the needs for predictability.

Finally, an appropriate event model is presented, as well as the COoperating Smart devices (COSMIC) middleware, which reflects the properties of GEAR and allows specifying events with attributes to express spatial and temporal properties. This is complemented by the notion of *Event Channels (EC)*, which are abstractions of the underlying network and enforce the respective quality attributes of event dissemination. Event channels reserve the needed computational and network resources for highly predictable event systems.

The paper is organized as follows. Section 2 presents related work. Then, Section 3 introduces fundamental notions and abstractions related to the sentient object model adopted in this paper. GEAR is then described in Section 4, while Section 5 describes the event model and the COSMIC middleware, which may be used to specify the interaction between sentient objects. The temporal aspects of interactions, which are crucial in the context of this paper, are covered in Section 6. Then, Section 7 presents two examples of the applicability of GEAR concepts. Finally, Section 8 concludes the paper.

2 Related Work

Our work considers a wired physical world in which a very large number of autonomous components cooperate. They are interconnected by multiple heterogeneous networks. 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. In an earlier work, we referred to such a network structure as a Wide-Area-Network of Controller-Area-Networks (WAN-of-CANs) [4]. Due to the dynamic nature of interactions, it will be difficult or impossible to know a priori the communication participants and secondly, because of heterogeneity, we have to support a variety of different addressing structures and mechanisms. Thirdly, components should be autonomous and no control transfer should be bound to communication. In general, event-based systems supporting autonomy of components and the spontaneous dissemination of information have been recognized as appropriate to support large scale distributed systems [5, 6]. Autonomy is achieved by a data driven model in which components decisions and actions are based on sharing data rather than on explicit control transfer [7]. Dynamic interactions are supported by content and subject related addressing schemes that enable communication without a priori knowing addresses or names of communication participants. Intended for large general purpose distributed applications, systems like [5, 8, 3] require quite complex infrastructures and do not consider stringent quality aspects like timeliness and dependability issues. A comprehensive overview and taxonomy is provided in [6]. Some publisher subscriber communication systems have been proposed to be used in control applications. The Information Bus [9] and its realization by the TIBCO Rendezvous software [10] integrates some soft real-time features. E.g., it is possible to create several prioritized queues and explicitly associate event types with event queues but real-time programming can be not handled at the same abstract level as the event programming. The NDDS protocol [11] and its successors from RTI [12] have some real-time properties. Its architecture explicitly assumes the Ethernet as the underlying interface. Therefore, real-time can only be assured on a probabilistic basis. It means that the communication load must be known in advance and that deviations from the load hypothesis are minimally tolerated. Deadlines are specified only for subscriptions. Therefore message scheduling is supported only in the queues maintained in the receiver side.

In the area of large control systems, Autonomous Decentralized Systems (ADS) have been proposed [13]. They provide a shared data field which decouples producers of information and consumers which autonomously may retrieve (control) information from the data field. ADS is based on a quite complex content-based addressing scheme similar to Linda [14] and thus is difficult to run on resource constraint components as smart sensors and actuators.

In [2] 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, 16]. They are targeted for the CAN- Bus [17], 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 [18] and has lead to the proposal of a standard by the Object Management Group (OMG) [19]. Smart transducers are organized in clusters, connected to a CORBA system by a gateway. They form isolated subnetworks in which a special master node enforces the temporal properties. A CORBA gateway allows to access sensor data and to write actuator data by means of an interface file system (IFS). 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 behavior of a single cluster is rigorously defined, no model to specify temporal properties for cluster-to-CORBA or cluster-to-cluster interactions is provided.

It should be noted however that all the event-based systems discussed in this section lack the holistic view of an architecture which integrates the events of the environment with those of the system. This is a pre-condition to meet the fundamentally new consistency and correctness challenges brought by complex systems of embedded systems. In particular, we address the kind of synchrony properties that allow the ordering and the scheduling of the events on the communication medium, and the conditions for their enforcement.

3 Sentient Object Model

3.1 Information Flow and Interaction Model

We consider a component-based system model that incorporates previous work developed in the context of the IST CORTEX project [20]. A fundamental idea underlying the approach is that applications can be composed of a large number of smart components, which can sense and interact with their surrounding environment. They are referred to as *sentient objects*, a metaphor elaborated in CORTEX and inspired on the generic concept of *sentient computing* introduced in [21]. 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. The following kinds of interactions can take place in the system: (i) environment-to-object interactions, reporting about the state of the former, and/or notifying about events taking place therein; (ii) object-to-object interactions, complementing the assessment of each individual object about the surrounding space, or serving collaboration with other objects; (iii) object-to-environment interactions, with the purpose of forcing a change in the state of the latter.

The environment can be a producer or consumer of information while interacting with sentient objects, which it does through transducers: sensors and actuators. The latter perform the necessary transformations between the physical real-time entities and their computerized representations in the system [22]. In our architecture, there are two kinds of transducers: *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" the "dialect" of our event model. Transducers may, or may not be part of a sentient object's body, as discussed in Section 3.4.

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, through links established between actuations and sensing operations. Thus the environment constitutes an additional interaction and communication channel and is in the control and awareness loop of the objects. It has been shown that in systems ignoring these hidden channels (e.g., feedback loops) inconsistencies may arise that can lead to unexpected failures [22]. In order to deal with the global information flow through computer system and environment in a seamless way, handling "software" and "hardware" events uniformly, it is necessary to find adequate abstractions. As discussed in Section 4, the Generic-Events Architecture introduces the abstractions of *Generic Event* and *Event Layer* to deal with these issues.

3.2 Component-based Object Model

The approach proposed in this paper is based on a component-based object model that incorporates some of the ideas developed in the context of the CORTEX project. Applications are composed of a (possibly large) number of smart components that are able to sense their surrounding environment and interact with it, which are referred to as *sentient objects* [23] (see Figure 1).

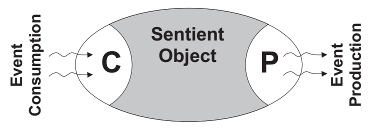

Fig. 1. The sentient object metaphor.

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.

3.3 Sentient Object Composition

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.

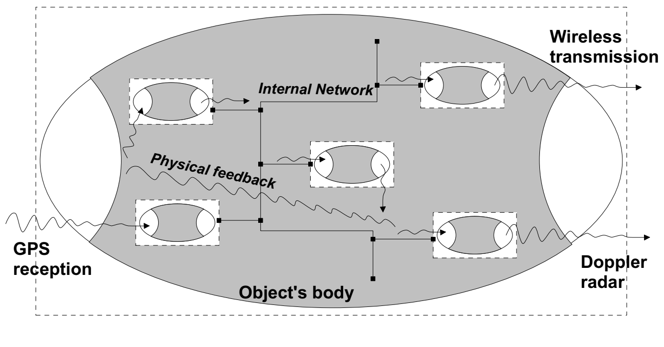

Fig. 2. Component-aware sentient object composition.

This is illustrated in Figure 2, 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.

To give a more concrete example of such *component-aware* object composition we consider a fully-fledged sentient object, for example a car. The car is composed of several sentient objects, like: WLAN receiver and transmitter processor; velocity sensor processor; cruise speed control processor and actuator; doppler radar control; GPS CCD camera input treatment modules; control elements such as cruise speed, platoon, ambient, visual display, etc. The car (together with all of its embedded software) is in turn a sentient object, and the environment internal to its own structure becomes this larger object's *body.*

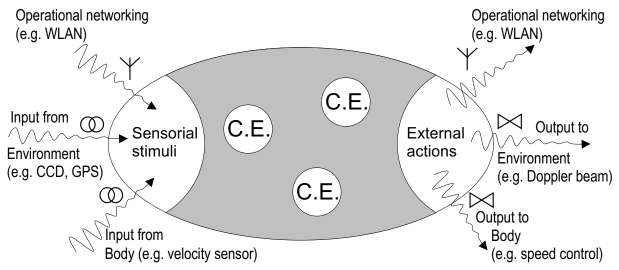
Operational networking (e.g. WLAN)
Input from Environment (e.g. CCD, GPS)
Sensorial stimuli
Input from Body (e.g. velocity sensor)
C.E.
C.E.
C.E.
External actions
Operational networking (e.g. WLAN)
Output to Environment (e.g. Doppler beam)
Output to Body (e.g. speed control)

Fig. 3. Information flow through a sentient object (C.E- control element).

Figure 3 shows this perspective. The car as an object receives events from various different sources, namely operational networks (e.g., WLAN receiver), remote sources (e.g., GPS receiver) or local sources (e.g. velocity sensor). Likewise, it produces events to be consumed by different sinks, for instance events transmitted through networks (e.g., WLAN transmitter) to the environment or other objects, events to remote sinks (e.g., doppler radar actuator) or events to local sinks (e.g., speed control actuator). At this level of abstraction, it should be possible to define cooperation activities among several cars as sentient objects (say, platooning) without the need to know the internal structure of cars, or the events produced by body objects or by smart sensors within the body. Note that interactions within the local scope are referred to as interactions with the *body* of the object. This concept will be developed in the next section.

3.4 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 mobile units, 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 fieldbus like CAN [24, 17], TTP/A [18], LIN [25] or FlexRay [26], 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 [4] 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 examples, a vehicle (robot or car) 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 2). 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 Section 4, the notion of an *event channel* represents this construct in our architecture.

4 Generic-Events Architecture

Although literature has classically studied the networking and sensing/actuating problems in isolation, we propose the innovative concept of *generic event*, be it derived from the boolean indication of a door opening sensor, from the electrical signal embodying a network packet (at the WLAN aerial) or from the arrival of a temperature event message.

Likewise, the programs running in sentient objects have very often consistency requirements that derive, even if remotely, from what are called *real-time entities*, in fact representations of state variables of the surrounding environment. Some of these, referred to as *time-value entities*, have consistency conditions based on the timeliness of the operations controlled by the computer, vis-a-vis their evolution in the environment (e.g., for the cooling system to consistently use the temperature of the engine it must obey some timeliness constraints) [22].

To fulfil this vision, we require an event model that satisfies these two sets of requirements. On the one hand, a model that treats the information flow through the whole computer system and environment in a seamless way, handling "software" and "hardware" events in a generic way. On the other hand, one that allows defining global, end-to-end, non-functional criteria in the time domain, such as temporal consistency, or QoS guarantees. We address these issues in this section and following ones.

We propose the **Generic-Events Architecture (GEAR)**, depicted in Figure 4, which we briefly describe in what follows (for a more detailed description please refer to [27]). The unusual L-shaped structure is crucial to ensure some of the properties described.

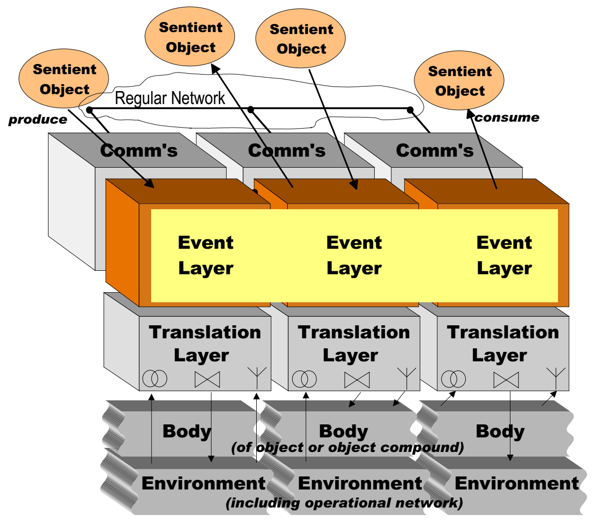

Fig. 4. Generic-Events architecture.

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

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.

Translation Layer: The layer responsible for physical event transformation from/to their native form to event channel dialect (in Section 5 we further elaborate on the definition of the events flowing through event channels), between environment/body and an event channel. This layer performs observation and actuation operations on the lower side (as represented by the sensor and actuator symbols in the figure), and 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.

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 5.

The GEAR architecture serves an object model based on production and consumption of generic events. Events are presented to objects through Event Channels (EC), which are in charge of propagating them to the relevant objects (those having subscribed to that event class). However, not only objects produce or consume events, but this is transparent, since it is dealt with by ensuring all these entities (objects and other) speak the EC dialect.

Events are produced by several sources which are treated in a homogeneous way. Event sources include:

- the environment where physical events take place, such as the detection of the opening of a gate, or of the change of a semaphore light, the sampling of a temperature at a given time.
- the body of the object (or object compound), taken as that part of the environment which is aggregated to the object or object compound and would not make sense otherwise (e.g., the body of a robot, the hardware of a car, the embodiment of a mechatronic device), and where similar kinds of physical events take place, for example, the sampling of a car's velocity. Note that part of what is 'environment' for an isolated object, may become 'body' of a compound object of which that object is part.
- the objects themselves may generate an event, when they invoke *produce*, which manifests itself as: a piece of information or a command they wish to make available to other objects; a notification they produce "to whom it may concern"; an actuation command on the body or on the environment, for example, controlling the speed of a car, or telling a gate to close.

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, as illustrated in Figure 5, 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 (they deserve the 'smart' adjective because they speak the EC dialect). 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

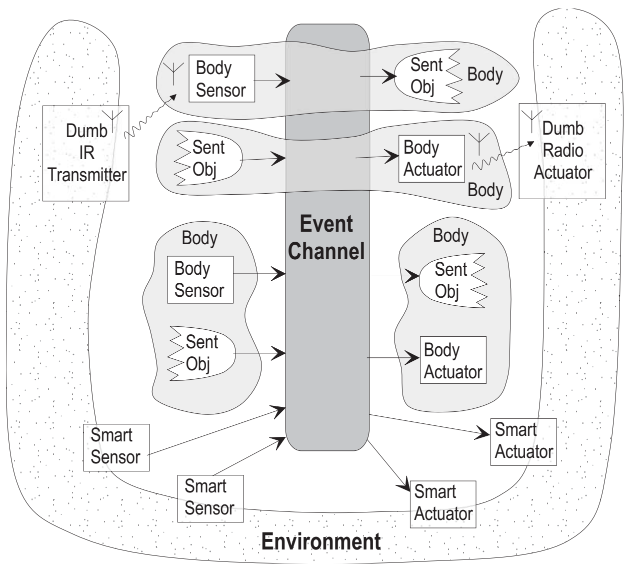

Fig. 5. Information flow in the whole system.

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 computer message passing and physical feedback loop information flows. We address this issue in Section 6.3.

5 Event Model and Middleware

An event model and a middleware suitable for smart components must support timely and reliable communication and also must be resource efficient. The **COSMIC (COoperating Smart devices)** middleware, which is described ahead in this section, is aimed at supporting the interaction between those components according to the concepts introduced so far. However, we first introduce the event model.

5.1 Event Model

Based on the WAN-of-CANs model, we assume that components, smart sensors/actuators or sentient objects, are connected to some form of CAN as a fieldbus or a 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 generic event model.

We introduced the notion of an event channel to cope with differing properties and requirements, and to have an object to which we can assign resources and reservations, as needed when striving for predictability. Although the concept of an event channel is not new [2, 8], 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 which may be issued to the channel. Every event subject is therefore associated to its own event 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 supports this approach. Our goal is to handle the temporal specifications as ⟨*bound, coverage*⟩ pairs [28], which is orthogonal to the more technical questions of how to achieve a certain synchrony property of the dissemination infrastructure. Exception handlers can be provided to deal with situations such as the violation of specified timeliness bounds for the channel.

Events are typed information carriers and are disseminated in a publisher/ subscriber style [9, 29], which is particularly suitable because it supports generative, anonymous communication [14] 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 [9, 29, 7] and crucial properties to maintain autonomy of components and dynamic interactions.

Events 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:

$$event := \langle subject, context\_attributeList, quality\_attributeList, contents \rangle$$

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. These timeliness, and other temporal aspects of the interactions will be further addressed in Section 6.

5.2 COoperating Smart devices Middleware

The COSMIC (COoperating Smart devices) 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 [30].

Correspondingly, this allows us to distinguish three event channel classes with different 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 [31], TTP/A [18], and TTCAN [24]. 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 [30].

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 to prevent 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 [32].

On the architectural level, COSMIC distinguishes three layers roughly depicted in Figure 6. 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 [7]. 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 [32], we describe the protocols and services of the abstract network layer particularly for the CAN-Bus.

As can be seen in Figure 6, the hard real-time (HRT) message class is supported by a dedicated handler which is able to provide the time triggered message dissemination. 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

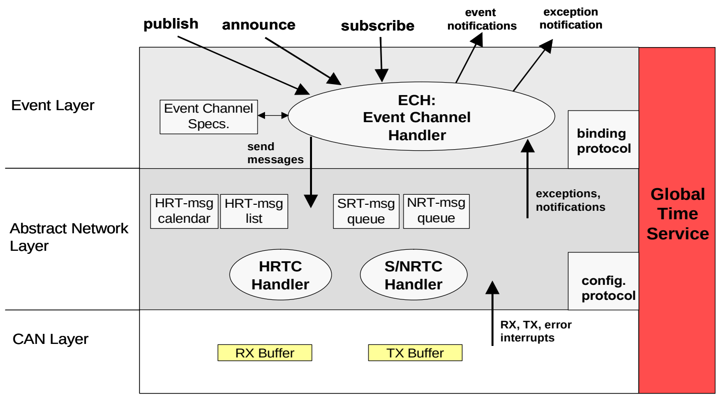

Fig. 6. Internal structure of COSMIC.

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.

6 Temporal Aspects of Interactions

Any interaction needs some form of predictability. If safety critical scenarios are considered, 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 a 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 [33, 28].

In most real-time systems, the notion of a deadline is the prevailing scheme to express and enforce timeliness. However, firstly, a deadline only weakly reflects the temporal characteristics of the information which is handled. In a dynamic cooperative control system suffering from weak links it is important to consider issues like ageing of information or the urgency and importance of control actions, which is not possible within the notion of a simple deadline.

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 an 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 necessarily be reverse engineered from a deadline.

Thirdly, as is emphasized in the work of Mock [34], the notion of a deadline assumes a passive environment and an active control system. In this classical view, the physical properties of the environment define the real-time constraints which are imposed as deadlines on a control system. This simple distinction in a passive environment and an active control system does not reflect the situation in a cooperative scenario where multiple active and mobile entities interact. The cooperating entities form part of the environment perceived by an individual component and they are active. Consider a team of robots which cooperatively move or lift an object too heavy for a single robot. In such a coordination task where mutual dependencies exist, the notion of a deadline is hardly useful when timing the interaction.

Therefore, we need a more elaborated model of representing and exploiting temporal information in such an environment. To derive such a model, we briefly review the notion of events in the physical environment and their representation in a machine.

6.1 Events

In the physical world, an **event** is defined as a singular occurrence in space and time. It can be defined as a cut of the time line and point in space and thus has no temporal or spatial extension. An event may lead to a state change in the physical world which may be called an effect. The effect is the subject of an observation [35, 36, 22]. The observation has a temporal and spatial distance to the related event. In a dynamic system, it is beneficial to store and transfer information about the temporal and spatial occurrence of an event together with the event itself because implicit assumptions about where and when the event occurred may be difficult. Information related to these issues is called the event context. In GEAR, a **generic event** is a happening that takes place in the event layer at a given instant of the time line, $\langle E, T^{GE} \rangle$. The happening is internal to the system, has an event channel compliant representation, and may be related with physical events taking place in the environment. That is, the 'event' is the happening as seen by the event layer, at a given instant in the time line.

It is useful to consider the event model in the terms defined in [37, 35, 36, 22], i.e. a real-time entity (RTe), which is the element of the physical world, a real-time image, which is a representation of the environment and can be a single image of a RTe or a complex representation derived from multiple sensor readings, and a real-time object, which refers to the data structure in which a real-time image or a real-time entity are stored and communicated. A real-time image derived from one or more observations may be encapsulated in a ⟨*context, value*⟩ pair, where context describes relevant aspects of the environment/system state in which the value has been derived. The classical ⟨*time, value*⟩ model of an event therefore refers to the temporal aspect of the context only. It is useful in a dynamic system as envisaged in this paper to include more information, e.g. location information or a network zone.

6.2 Temporal properties of events

In order to deal with real-time sentient objects we need to understand the implications of timeliness requirements in the context of the proposed generic-events architecture. This will be done by establishing fundamental correctness criteria for the operation of the system. The system architecture, including the protocols and mechanisms materialising the event layer middleware, must be built so that the strict observation of the established criteria is ensured. On the other hand, given the distributed nature of the problem, the correctness of operation does not depend solely on the observation of timeliness constraints, but also on the consistency and coordination among the distributed actors in the system. In this respect, note that the information flow is defined in terms of events, and it is controlled at the event layer, where everything passes. As such, and very importantly, all consistency criteria that must be secured apply as well to regular messages, messages through operational network channels, and input/output feedback paths through the environment. No hidden channel problems need affect the operation of the system [22].

We illustrate the nature and the temporal properties of generic events with a few examples.

Examples of generic events

1	door opened;
2	door opened as observed at T;
3	door is open;
4	door is open as observed at T;
5	temperature is X;
6	temperature is X as observed at T;
7	position of crankshaft is Y;
8	position of crankshaft is Y as observed at T;
9	crankshaft reached ignition point I;
10	crankshaft reached ignition point I as observed at T;
11	value of variable Z is 'entering-zone mode';
12	set variable 'cruise speed' to S;
13	set variable 'cruise speed' to S within T;

The difference between (1) and (2) is that in (1) we know at T^{GE} that the door has opened in some (near) past instant, whereas in (2) we know at T^{GE} that it opened at T. The difference of the former to (3) and (4) is that here we know the state of the door, without necessarily knowing when it opened. In fact, note that generic events report state (3) as well as change of state (1), what in other more classical models used to denote "state" and "events", with regard to the physical environment. This said, in GEAR nothing prevents the periphery of the system (e.g., smart sensors) from being organized in the best suited way (e.g., state sampling, event latching, etc.), for each generic-event flow to be produced.

Given that T^{GE} establishes the event production instant, there is apparently redundant timing information in some lines, e.g. (2) or (4). However, this is an important characteristic of GEAR: T^{GE} denotes the time at which E was produced on the event channel and serves any generic type of event; T is part of E, thus invisible to the EC, but it denotes the time at which a given real-time entity was observed to have a given state or to have changed its state. The separation of concerns enforced by T and T^{GE} is very important, as we detail ahead.

When we say in (11) that we know at T^{GE} that " Z = 'entering-zone mode' ", this marks the point at which this internal state change is relevant for the system, e.g., as alerted by the platoon leader sentient object, in a cooperating cars scenario. Likewise, in (12), when (e.g., the leader again) publishes the command to change the state of the 'cruise speed' variable to S, the reference point is T^{GE}.

However, when we say in (3) that we know at T^{GE} that "the door is open", or in (5) that "the temperature is X", we might as well try to know how trustworthy this information is, since the temperature and door are time-value entities. For example, by the time T^{GE} when we learn that "the temperature is X", it might already be way higher than X! Even if, as we say in (4), we know at T^{GE} that "the door is open at T", or as in (6), that "the temperature is X at T", this still may not solve our problem. There are important implications of the way we handle time-value entities that we discuss below, using definitions in [22].

Firstly, saying, as in (6), that we know at T^{GE} that "the temperature is X at T", might seem to provide a precise indication. However, what T portrays is the time at which the periphery of the system observed the temperature. When observing the value of a continuous variable, it is relevant to define the error. For an observation $\langle r(E_i)(t_i), T_i \rangle$ of the value of an RTe E_i at t_i receiving timestamp T_i, the **observation error** in the *value domain* is given by $\nu_i = |E_i(T_i) - r(E_i)(t_i)|$: we expect the value of E_i at T_i, but we get an approximation of the value ($r(E_i)$), measured approximately (t_i) at T_i. That is, the error has two components, the inaccuracy of the sensing apparatus, and the observation positioning error. On the other hand saying, as in (4), that we know at T^{GE} that "the door is open at T", has analogous constraints. Here, when observing the time at which a given discrete value E_i occurs (e.g., opening of the door), the observation error (jitter) is defined in the *time domain*, $\zeta_i = |T_i - t_i|$: E_i assumed a given value at t_i, but the system logs it as having happened at T_i.

In order for our measurements to be useful, we establish bounds on the observation errors, for classes of events produced in certain conditions (e.g., that maxima of the sensor errors and of the clocks precision are known for that class). The fact that observations of a class of events meet an error bound allows us to abstract from measurement details and henceforth trust observations of that class. We say they are *consistent*:

- Given a known $\mathcal{V}_o$, we say that an observation is **consistent** in the **value** domain, if and only if $\nu_i \leq \mathcal{V}_o$
- Given a known $\mathcal{T}_o$, we say that an observation is **consistent** in the **time** domain, if and only if $\zeta_i \leq \mathcal{T}_o$

This deals with the consistency at the observation instant. Secondly, we must deal with the consistency at the use instant. We must ensure that the information is sufficiently fresh when it is about to be used. For example, when we say in (5) that we know at T^{GE} that "the temperature is X", it is important that the interval between the time when it was measured and T^{GE}, is known and short enough to be useful, so that the temperature hasn't drifted too much in the meantime. I.e. for the information provided by this generic event to be meaningful for whatever the system intends to do with it. This must be ensured by the infrastructure, and a practical way is to define a fixed parameter, known at design time, based on estimates of the variable's dynamics. This interval has been called *absolute validity interval* for databases [38], or *temporal accuracy interval* for control [36].

In fact, in GEAR we generalise this concept and we use the following two important notions to relate the value domain and the temporal domain of information: temporal consistency and temporal validity.

Temporal consistency relates a bound ν in the value domain to a point in time t. In general, since we consider real-time entities, the value v of the RTe is dependent on the time and can be described as a function over time (value over time [22]): $f_c(t) = v$. For a point in time $t_a > t$, the equation becomes $f_c(t_a) = v + \delta_a$ where δ_a is the change of the value in the interval $[t, t_a]$. Temporal consistency defines a bound ν on the value v at time t_a for which v is an acceptable representation of the RTe. The time-value entity is temporally consistent at t_a if and only if $0 \leq |\delta_a| \leq \nu$.

Temporal validity considers f_c and defines a time interval during which all values v of the time-value entity can be considered to be temporally consistent. Thus, given the temporal validity interval $[t_a, t_b]$ and a function $f_c(t)$, for any t_c, $t_a \leq t_c \leq t_b : f_c(t_c) = v + \delta_c \;\; (0 \leq |\delta_c| \leq \nu)$.

Temporal consistency specifies how close an observed RTe captured in a time-value entity represents the value in the real world. Temporal validity intervals are derived from the knowledge of both the temporal consistency requirements and the function f_c. Both, temporal consistency and temporal validity can be used to reason about properties of an event independently of any implementation issues, thus they should be part of a specification. Figure 7a) shows the simple case when $f(t)$ is a linear function. Temporal consistency of a value v is defined by a unique interval which matches the temporal validity interval for t_0 as observation point.

Figure 7b) highlights the fact that temporal consistency is independent from a temporal validity interval. There is an implication in only one direction: A value in the temporal validity interval is always temporally consistent. However, the reverse is not true: A temporally consistent value may exist outside of a temporal validity interval. Note that if $f(t)$ is known, it may be possible to make predictions about the error and choose the right points in time for values bounded by ν just by a clock.

Figure 7c) shows a difficult case of a discrete function. It shows a discrete binary function which represents e.g. the state of a switch, a door, a traffic light,

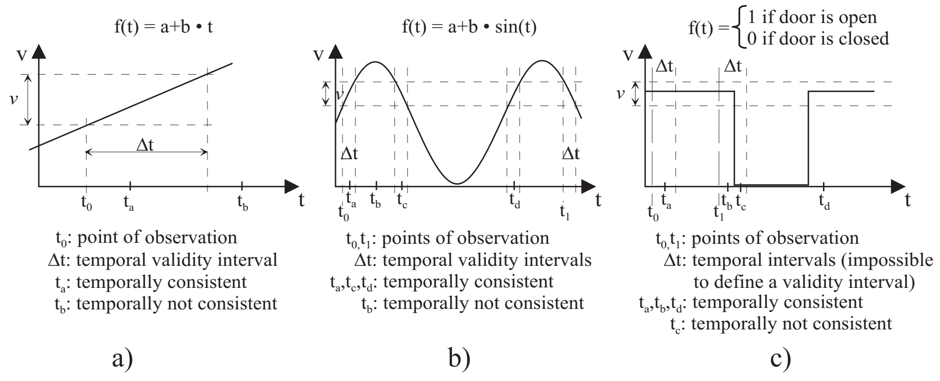
f(t) = a+b • t
v
v
Δt
t
t0 ta tb
t0: point of observation
Δt: temporal validity interval
ta: temporally consistent
tb: temporally not consistent
a)
f(t) = a+b • sin(t)
v
Δt
Δt
t
t0 ta tb tc td t1
t0,t1: points of observation
Δt: temporal validity intervals
ta,tc,td: temporally consistent
tb: temporally not consistent
b)
f(t) = 1 if door is open / 0 if door is closed
v
v
Δt
Δt
t
t0 ta t1 tb tc td
t0,t1: points of observation
Δt: temporal intervals (impossible to define a validity interval)
ta,tb,td: temporally consistent
tc: temporally not consistent
c)

Fig. 7. RTe behaviours: a) Linear monotonic function; b) Continuous cyclic function; c) Discrete function.

etc. It is very difficult to define temporal validity intervals because the function is not steady. How small the interval Δt ever is, it may include a rising or falling edge of the function and therefore a temporal inconsistency. Although temporal consistency can be easily specified, no temporal validity interval can be defined.

Finally, coming back to our examples of generic events, the difference between (1,2) and (5,6) concerns the nature of the variable: discrete in the former, and continuous in the latter. Lines (7-10) illustrate how this distinction, so much used in computer control, may turn out to be pretty much artificial. The position of an engine's crankshaft is a continuous variable: an angle that goes from 0 to 360 degrees (0 again) and so forth. So, there is apparently no difference between (5,6) and (7,8). However, the crankshaft evolves so quickly that addressing it as a continuous variable may imply a very high error. Hence, if we "fabricate" a discrete variable which is the arrival of the crankshaft to the ignition point I, as in (9), then this is equivalent to the kind of event in (1). This ambiguity was addressed in [22] as the duality between *value over time* and *time of a value*.

In conclusion, we have shown the fundamental consistency guarantees to be ensured by this kind of architectures: *value* and *time* domain observation *consistency*; and *temporal consistency*.

6.3 Event ordering and the hidden channel problem

As mentioned before, one important problem in real-time computer control and embedded systems is the inconsistency between computer message passing and physical feedback loop information flows. This stems from the difficulty in ensuring a proper ordering of event-messages (exchanged among computers in the system) and physical events (which may propagate through the environment, establishing precedence relations not perceived by the system). This is also referred as the problem of *hidden channels*, which we briefly introduce and discuss in this section. Let us consider a classical example that was given by Kopetz and Veríssimo [35] long time ago (see Figure 8).

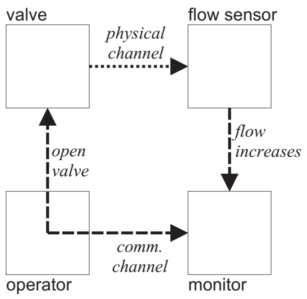

Fig. 8. Example of interaction through the environment.

The operator sends a message to open the valve over the network. At the same time she informs the monitor to be prepared that the flow in the pipe will change. This message is delayed and thus, when the flow sensor reports an increased throughput, the monitor infers a broken valve and erroneously shuts down the system. The communication via the physical (hidden) channel between the valve and flow sensor is faster than the worst case delays in the network.

Now, what is the interpretation of this example in terms of the event architecture? The event layer supports multiple channels and channel classes to disseminate event messages. Event messages have a temporal validity, from which transmission and delivery deadlines are derived. A possible application design is depicted in Figure 9a). The command from the operator is not a safety critical event and thus, it is pushed to a soft real-time channel. The valve and the monitor have subscribed to this channel. The flow sensor may publish throughput periodically to a hard real-time channel because the monitor has to react on flow changes with a guaranteed latency. The problem with the scenario is that because of the interaction through the physical channel, it is not possible to apply one of the usual ordering schemes based on some history mechanism because there is no gap detection property. The physical channel simply cannot carry any ordering information. The problem is how to detect the causal relationship between the event which caused the valve to open and the subsequent message of the flow sensor. Any attempt to solve this problem in an asynchronous system will fail. We need some mechanism based on time to detect the (potential) causal dependency between the event generated by the flow sensor and the previous command of the operator.

The dissemination of events through the event layer alone does not help much if we only consider the communication over the regular network. The question is how to include the physical channel? Figure 9b) sketches the situation.

We recognize that we have a physical (hidden) channel between the valve and the flow sensor. This channel has a known latency. Within the GEAR architecture, it can be interpreted to be propagated via a kind of operational network coupling the valve and the flow sensor. The physical property of a flow speed is transferred through the translation layer of GEAR by the smart flow sensor. At the event layer it is provided as a periodic event propagated through a hard

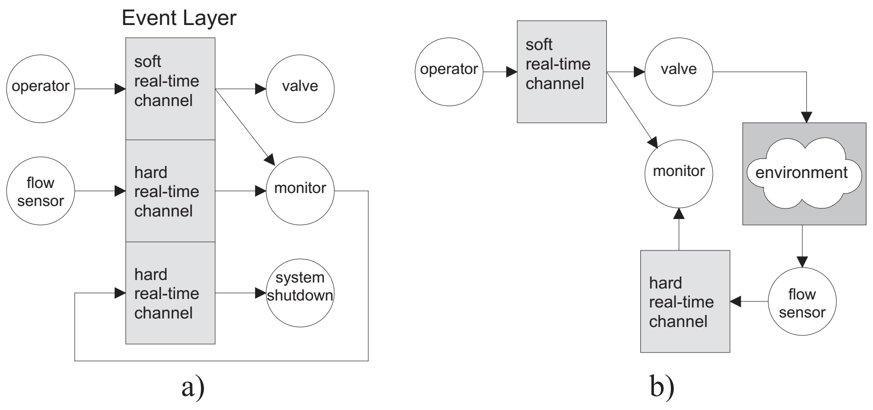

Fig. 9. The example modelled in an event system: a) without physical channel; b) including the physical channel.

real-time channel. So far, no ordering property is available in the system. The monitor can not decide whether the event from the flow sensor is dependent on a soft real-time command event previously published by the operator. There is no gap detection property. We need a synchrony property to solve this problem.

We address the problem using the theory developed in [39]. Hidden channels are not directly controlled, instead their effect is incorporated in the design of the regular computer communication system. The latter is endowed with the adequate synchronism so as to guarantee that hidden channels do not cause ordering inversions. This is akin to designing a real-time causal delivery messaging system sensitive to both computer (send, receive) and real-world (sense, act) events. The communication system is tuned with the propagation time parameters of the environment.

Given the above, the problem can be addressed by resorting to a solution based on a model of partial synchrony, such as the TCB [28]. We make the assumption that we have a network where we can specify ⟨bound, coverage⟩ pairs and have best effort mechanisms to enforce deadline delivery for soft real-time events. The TCB provides awareness in the case an event misses the deadline or is delivered in a different order at different nodes. It doesn't make sense to defer a hard real-time event to provide a global order between hard real-time and soft real-time events. In other words, the event from the flow sensor is not deferred until the (soft real-time) command event has been delivered to the monitor. Thus, the HRT event from the flow sensor may be delivered before the soft real-time command event. However, this would be known to the monitor, which, with the help of the TCB, would be able to detect when an event has not been delivered within a certain time bound. The latency of the entire chain of event dissemination as depicted in Figure 10 would be used to set the TCB to an appropriate value (the minimum propagation time of the event chain, including the propagation time through the physical channel).

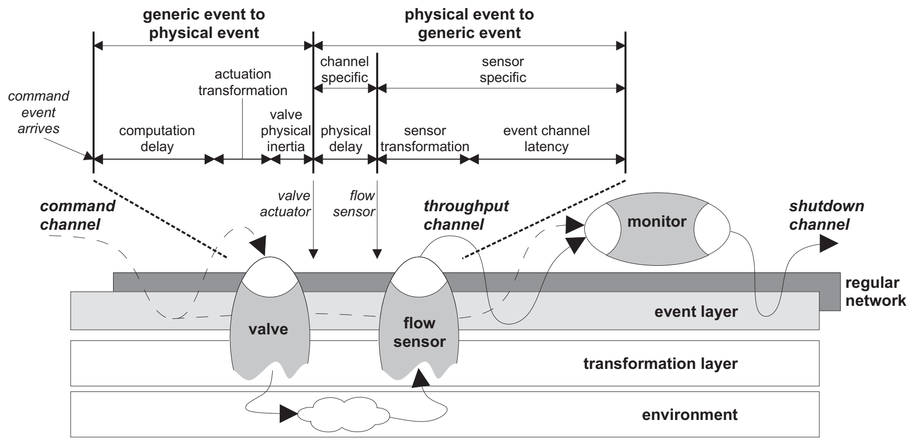

Fig. 10. Temporal analysis of event flow.

Considering the flow sensor as a sentient object, it consumes the physical event from the environment generated by the actuation of the valve. In turn, it generates a generic event pushed to the respective channel. The latency can be calculated easily from the latency of the physical channel and the HRT channel of the event layer. At the abstraction level of sentient objects, we have three sentient objects and two event channels, one disseminating the commands and one disseminating the flow in the pipe (throughput) measured by the flow sensor. Figure 10 show this in different detail.

The main benefit including the physical channel would be the fact that now the bounds for the latencies as needed for setting the TCB is explicitly stated in the design phase.

7 Practical examples

This section addressed two short examples of proof-of-concept prototypes that have been developed in the course of the CORTEX project, in which we used some of the concepts elaborated in this paper. A more detailed description of the cooperating cars example is provided in [40], while the coordinated robots example and the COSMIC middleware are further detailed in [41].

7.1 Cooperating Cars

In the first scenario we consider several cars (sentient objects) with the ability to 'consume' events from the environment and 'produce' events to it. These events are disseminated and received through Event Channels, using the GEAR architecture. We assume that the cars can freely move around and, for simplicity, we do not consider obstacles. Because of that, a fundamental safety rule consists in ensuring that no car crashes occur, which requires every car to know the position of other cars. Therefore, each car periodically publishes its position as a GEAR event, which will be consumed by the cars that subscribed it and that fulfil the conditions to receive it (e.g. they are in the same 'zone').

One solution to satisfy the safety rule is to provide each car with a temporally consistent image of the external and internal (body) environment. Smart sensors publish events about the relevant real-time entities of the environment. Body sensors do the same with respect to the body. These events define the state of a 'zone' surrounding the car. Now, suppose all sentient objects feature a module, call it constructor, in charge of building a *real-time image* (RT-image) of the zone. The constructor subscribes to the environment and body events, and maybe other relevant events. The external environment events contribute to form a public RT-image of the zone. That image is enriched with inputs from other objects, and from the object's own body, part of which may not be made public. What is important is that *all* these events obey global consistency rules.

The safety rule is illustrated on the left of Figure 11. The grey car must keep outside the dashed circles around the black cars, and so he must always know the position of the black cars with a bounded error (ϵ). This error depends on how much time has passed since the last position of the car was published and on the maximum speed of that car. Therefore, both of them must be bounded.

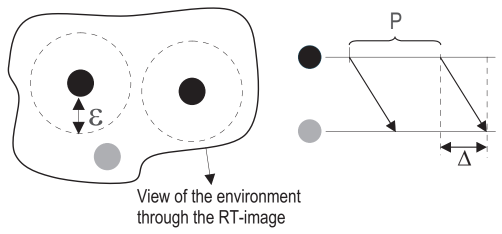

Fig. 11. Safety rules in the cooperating cars scenario.

The problem would be easily solvable if we could assume a reliable and timely propagation of events through EC modules. The system would be configured so that at every P time units the grey car would receive information from the black car (see Figure 11 on the right). Every car would be able to construct a RT-image of the surrounding cars with a bounded accuracy (related with the propagation latency, Δ, and the period, P), and all the images would be consistent among them. However, since we have to assume asynchronous Event Channels (the abstract network layer is not able to provide real-time channels in the operational environment, typically wireless, considered here), it is not possible to ensure the temporal consistency of the RT-image. Events can take more than Δ to be propagated, incurring in timing failures.

In this example we assume that the architecture, despite uncertainty of communication, is endowed with perfect timing failure detection (TFD), such as supported by a timely computing base (TCB) [28]. This allows the construction of event-channel handlers that, despite not being able to provide real-time properties, are able to provide indications when some specified timeliness bound is violated. This kind of EC modules can be exploited by fail-safe applications, such as the cars we consider here, which can stop as soon as a timing failure occurs. Considering our cooperating cars scenario, the required timeliness bounds would have to be specified as quality parameters of the EC modules. The EC module would then observe the timeliness of the events, with the help of the TFD service, and execute a fail-safety procedure to stop the car, in case of a timing failure.

7.2 Coordinated robots

The coordinated robots example focuses on a demo of two cooperating robots, depicted in Figure 12. Each robot is equipped with smart distance sensors, speed sensors, acceleration sensors and one of the robots (the "guide" (KURT2) in front (Figure 12)) has a tracking camera allowing to follow a white line.

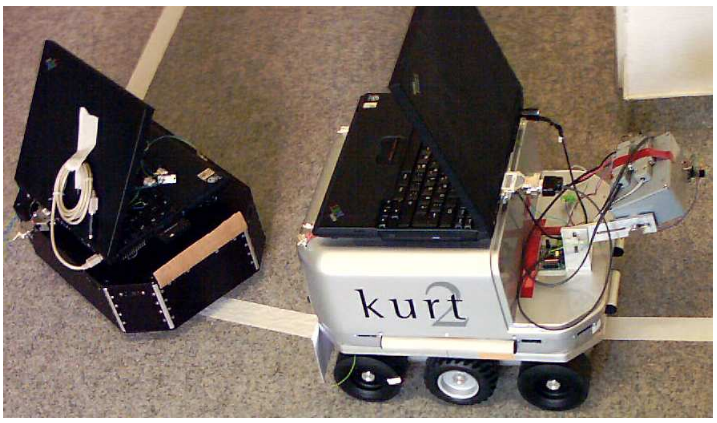

Fig. 12. Cooperating robots.

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

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.

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.

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.

8 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, in essence, systems-of-embedded-systems. We cannot assume that the network architecture in such a system is homogeneous. This work is a contribution towards achieving seamless integration of different components in such an environment, controlling 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 interactions through the environment and the network. To our knowledge, it is the first architecture to provide the possibility for hidden channel avoidance in the model, that is, a seamless integration of physical and computer information flows. This may have important implications on the way to architect systems-of-embedded-systems.

The notion of event channel has been introduced which allows to specify quality aspects explicitly, namely temporal properties. 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.

We believe the concepts elaborated here to be of interest to build innovative event-based systems that have to portray some kind of real-time behaviour, because they deal with the environment. This serves application areas such as: small-scale embedded systems; federated large-scale embedded systems; ambient intelligence settings; ad-hoc and mobile dynamically configurable systems.

References

1. Hightower, J., Borriello, G.: Location systems for ubiquitous computing. IEEE Computer **34**(8) (aug 2001) 57–66
2. Harrison, T., Levine, D., Schmidt, D.: 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), Atlanta, Georgia, USA, ACM Press (1997) 184–200
3. Meier, R., Cahill, V.: Steam: Event-based middleware for wireless ad hoc networks. In: Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02), Vienna, Austria (2002) 639–644
4. Casimiro (Ed.), A.: Preliminary definition of cortex system architecture. CORTEX project, IST-2000-26031, Deliverable D4 (April 2002)
5. Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O., Spiteri, M.: Generic support for distributed applications. IEEE Computer **33**(3) (2000) 68–76
6. Meier, R., Cahill, V.: Taxonomy of distributed event-based programming systems. The Computer Journal **48**(5) (September 2005) 602–626
7. Kaiser, J., Mock, M.: Implementing the real-time publisher/subscriber model on the controller area network (CAN). In: Proc. 2nd International Symposium on Object-oriented Real-time distributed Computing, Saint-Malo, France (May 1999)
8. (OMG), O.M.G.: CORBAservices: Common Object Services Specification - Notification Service Specification, Version 1.0 (2000)
9. Oki, B., Pfluegl, M., Seigel, A., Skeen, D.: The information bus - an architecture for extensible distributed systems. Operating Systems Review **27**(5) (1993) 58–68
10. TIBCO: Tibco rendezvous concepts, release 7.0 (April 2002)
11. Parado-Castellote, G., Schneider, S., Hamilton, M.: NDDS: The realtime publish subscribe network. In: IEEE Workshop on Middleware for Distributed Real-Time Systems and Services. (1997) 222–232
12. RTI: Real-time inovations. network data delivery service http://www.rti.com.
13. Mori, K.: Autonomous decentralized systems: Concepts, data field architectures, and future trends. In: Int. Conference on Autonomous Decentralized Systems (ISADS93). (1993)
14. Carriero, N., Gelernter, D.: Linda in context. Communications of the ACM **32**(4) (April 1989) 444–458
15. Kim, K., Jeon, G., Hong, S., Kim, T., Kim, S.: Integrating subscription-based and connection-oriented communications into the embedded CORBA for the CAN Bus. In: Proc. IEEE Real-time Technology and Application Symposium. (May 2000)
16. Lankes, S., Jabs, A., Bemmerl, T.: Integration of a CAN-based connection-oriented communication model into Real-Time CORBA. In: Workshop on Parallel and Distributed Real-Time Systems, Nice, France (April 2003)
17. Robert Bosch GmbH: CAN Specification V2.0. Technical report (September 1991)
18. Kopetz, H., Holzmann, M., Elmenreich, W.: A Universal Smart Transducer Interface: TTP/A. International Journal of Computer System, Science Engineering **16**(2) (March 2001)
19. (OMG), O.M.G.: Smart transducer interface, initial submission (June 2001)
20. CORTEX Consortium: CORTEX project Annex 1, Description of Work (October 2000) http://cortex.di.fc.ul.pt.
21. Hopper, A.: The Clifford Paterson Lecture, 1999 Sentient Computing. Philosophical Transactions of the Royal Society London **358**(1773) (August 2000) 2349–2358

22. Veríssimo, P., Rodrigues, L.: Distributed Systems for System Architects. Kluwer Academic Publishers (2001)
23. Meier (Ed.), R.: Preliminary definition of cortex programming model. CORTEX project, IST-2000-26031, Deliverable D2 (March 2002)
24. Führer, T., Müller, B., Dieterle, W., Hartwich, F., Hugel, R., M.Walther: Time triggered communication on CAN (2000) http://www.can-cia.org/can/ttcan/fuehrer.pdf.
25. LIN Consortium: Local Interconnect Network: LIN Specification Package Revision 1.2. Technical report (November 2000)
26. FlexRay Consortium: FlexRay Communications System Protocol Specification, Version 2.0. Technical report (June 2004)
27. Veríssimo, P., Casimiro, A.: 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 (January 2003)
28. Veríssimo, P., Casimiro, A.: The Timely Computing Base model and architecture. Transactions on Computers - Special Issue on Asynchronous Real-Time Systems **51**(8) (August 2002) 916–930
29. Eugster, P.T., Felber, P., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. Technical Report DSC ID:200104, EPFL, Switzerland (2001)
30. Livani, M., Kaiser, J., Jia, W.: Scheduling hard and soft real-time communication in the controller area network. Control Engineering **7**(12) (1999) 1515–1523
31. Kopetz, H., Grünsteidl, G.: 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)
32. Kaiser, J., Mitidieri, C., Brudna, C., Pereira, C.: COSMIC: A Middleware for Event-Based Interaction on CAN. In: Proc. 2003 IEEE Conference on Emerging Technologies and Factory Automation, Lisbon, Portugal (September 2003)
33. Becker, L.B., Gergeleit, M., Schemmer, S., Nett, E.: 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 (September 2003)
34. Mock, M.: On the real-time cooperation of autonomous systems. Fraunhofer Series in Information and Commnication Technology **6**(2204) (2004)
35. Kopetz, H., Veríssimo, P.: Real-time and Dependability Concepts. In Mullender, S.J., ed.: Distributed Systems, 2nd Edition. ACM-Press. Addison-Wesley (1993) 411–446
36. Kopetz, H.: Real-Time Systems. Kluwer Academic Publishers (1997)
37. Kopetz, H., Kim, K.H.: Temporal uncertainties in interactions among real-time objects. In: 9th Symp. on Reliable Distributed Systems (SRDS-9), Huntsville, AL, USA, IEEE Computer Society Press (1990) 165–174
38. Ramamritham, K.: The origin of TCs. In: Proceedings of the First ACM International Workshop on Active and Real-Time Database Systems, Skovde,Sweden, Springer-Verlag (June 1995) 50–62
39. Veríssimo, P.: Causal delivery protocols in real-time systems: A generic model. Journal of Real-Time Systems **10**(1) (January 1996) 45–73
40. Martins, P., Sousa, P., Casimiro, A., Veríssimo, P.: Dependable adaptive real-time applications in wormhole-based systems. In: Proceedings of the 2004 International Conference on Dependable Systems and Networks, Florence, Italy, IEEE Computer Society Press (June 2004) 567–572
41. Kaiser, J., Brudna, C., Mitidieri, C.: Cosmic: a real-time event-based middleware for the can-bus. J. Syst. Softw. **77**(1) (2005) 27–36