Event-Driven Support of Real-Time Sentient Objects

Paulo Veríssimo and António Casimiro
{pjv,casim}@di.fc.ul.pt
Univ. of Lisboa, Portugal*

Abstract

The emergence of applications operating independently of direct human control is inevitable. Research on high-level models for this class of applications— e.g. on autonomous agents and distributed AI— has revealed the shortcomings of current architectures and middleware interaction paradigms. If we focus on complex real-time systems made of embedded components, that evidence is even more striking. Event models have shown to be quite promising in this arena, but they often lack one or several of the following key points: seamless integration with a programming model; adequate layer structure; and the provision of support for non-functional attributes, such as timeliness or quality of service, or security. This paper discusses infrastructural support to construct large-scale proactive applications based on the use of real-time sentient objects, and is specially devoted to the latter two points.

1 Introduction

The emergence of applications operating independently of direct human control is inevitable. In fact, with the increasing availability of technologies to support accurate and trustworthy visual, auditory, and location sensing [11] as well as the availability of convenient paradigms for the acquisition of sensor data and for the actuation on the environment [1], a new class of large-scale decentralized and proactive applications can be envisaged.

However, research on high-level models for this class of applications— e.g. on autonomous agents and distributed AI— has revealed the shortcomings of current architectures and middleware interaction paradigms. Dealing with highly dynamic interactions and continuously changing environments and, at the same time, with needs for predictable operation, is still an open challenge. If we focus on complex real-time systems made of embedded components, then even more stringent requirements have to be taken into account, namely to achieve distributed, safe and timely process control. In this context, the provision of adequate interaction paradigms is a fundamental aspect [2]. Typical characteristics of this class of applications, such as autonomy or mobility must be accommodated, while allowing the possibility to handle non-functional requirements like reliability, timeliness or security.

In contrast with the client/server or RPC based paradigms supported by current state-of-the-art object-oriented middleware [13, 8], event models have shown to be quite promising in this arena [9, 15], but they often lack one or several of the following key points: seamless integration with a programming model; architectures with an adequate layer structure; and the provision of support for non-functional attributes. This paper is specially devoted to the latter two points, in the context of the IST CORTEX project (http://cortex.di.fc.ul.pt), which is concerned with the infrastructural support to construct large-scale proactive applications based on the use of real-time sentient objects. CORTEX proposes an object-oriented programming model based on anonymous event-based communication [16]. In broad terms, the system model is composed of the *environment* and a set of *sentient objects*, which are capable of sensing the former and act on it, and cooperate with each other.

Now, how to architect such a system, namely how to define placement and composition rules for software components, when our objects are often embedded systems, or collections thereof, where the differences between hardware and software are sometimes subtle? That is, we no longer have the acquired principle that has guided modular and distributed systems design for years [10]: there are hosts, which we put where we want in the system, and there are software components, e.g. objects, which we put in some host, and which we can move, migrate, replicate, etc, with a desired property called *transparency.* The solution we propose is based on a *component-based object model* [7]. This breaks with the traditional separation between the software

*Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-FET-2000-26031 (CORTEX), and by the FCT, though the Large-Scale Informatic Systems Laboratory (LaSIGE).

and hardware perspectives, pointing to seeing objects as mixed hardware/software components, although it is obviously possible to conceive an object as a software-only component.

Another problem in the context of architecture is the handling of the information flow in the system. Here, we wish to break with the traditional real-time systems view, which makes a neat separation between the physical flow of information through the environment and the message-based flow inside the computer system— the input/output subsystem being the barrier between the two. We propose that the event model treats physical and computer event propagation in a uniform way, and discuss the resulting advantages. For that, we define an adequate event middleware architecture based on the notion of *generic events*, which we call the *Generic Events ARchitecture (GEAR)*.

This addresses the functional part of event handling, but is not enough. Non-functional properties are paramount in systems dealing with the environment, and in the paper we briefly discuss two questions which are particularly relevant in the context of sentient objects: *temporal consistency* as the key criterion for consistency of operations on *time-value entities*; and the *dependable* management of *end-to-end QoS*.

The paper is structured as follows. Next section describes a component-based object model that builds on some of the ideas proposed in the CORTEX project. Then, Section 3 introduces the GEAR architecture to support the described object model. In Section 4 we focus on an example application scenario, in which we exercise some of the concepts and solutions proposed in the paper, and discuss the temporal consistency and QoS issues.

2 A Component-based Object Model

The approach proposed in this paper to architecturally structure applications composed of sentient objects is based on a component-based object model that incorporates some of the ideas developed in the context of the CORTEX project.

CORTEX assumes that a new class of applications can be envisaged, which is composed of a (possibly 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*. The basic CORTEX object metaphore was presented in [6] (see Figure 1). *Sentient computing* established the generic concept, presented in [12], elaborated in CORTEX in the context of object components. Generically speaking, sentient objects are objects that 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.

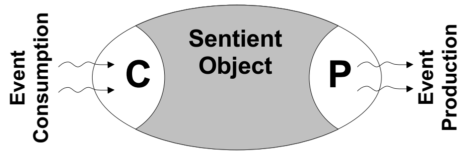

Figure 1. The sentient object metaphor.

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 to transform sensorial information received as input events, and output events in the course of it. 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 2, where a sentient object is internally composed of a few other sentient object, each of them consuming and producing events, some of which only internally propagated.

To provide an example of such *component-aware* object composition, we take a robot having as components the several axes and manipulator controllers: each of these controllers together with the control software may be a sentient object. The robot itself may then be a (composite) sentient object, composed of the objects materialized by the controllers, and the environment internal to its own structure, or body. The same reasoning can be applied to a car, with its internal components. The car body (together with its embedded software) may be a sentient object, composed of several simpler objects, like: WLAN receiver and transmitter card and driver; velocity sensor processor; cruise speed control processor and actuator; doppler radar control; GPS CCD camera input treatment mod-

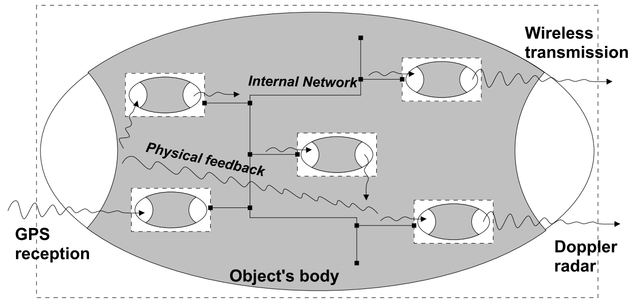

Figure 2. Component-aware sentient object composition.

ules; control elements such as cruise speed, platoon, ambient, visual display, etc. Note that in this system it makes no sense migrating the cruise control software to the ambient control hardware module— it should obviously be attached to the relevant control hardware. Distribution and location transparency/independency may still be desirable to a certain extent, but they must be addressed in the context of well-formed sentient objects that obey the component-aware structuring and composition approach.

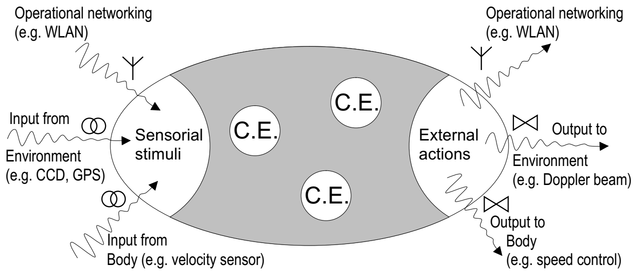
Operational networking (e.g. WLAN)
Operational networking (e.g. WLAN)
Input from Environment (e.g. CCD, GPS)
Sensorial stimuli
C.E.
C.E.
C.E.
External actions
Output to Environment (e.g. Doppler beam)
Output to Body (e.g. speed control)
Input from Body (e.g. velocity sensor)

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

Figure 3 shows the perspective of a fully-fledged sentient object, for example a car, receiving 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, the object 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). 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.

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.

In fact, what happens with classical event/object models is that they are software oriented. 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 fact, very often, the only "event" characteristic in "software" events is the arrival of the event-message itself (e.g., when it merely carries the state of a variable or an information to another object). If such classical desiderata of distributed systems such as distribution and location transparency/independency are to be realized to a certain degree, this conflict must be solved.

Furthermore, sentient objects deal with real-time aspects involving 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. 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) [18].

To fulfil this vision, we require an event model that satisfies these two sets of requirements, respectively of functional and non-functional nature. That is, a model that treats the information flow through the whole computer system and environment in a seamless way, handling "software" and "hardware" events uniformly. 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 the forthcoming sections.

3 Architecture

Given the component-based object model presented above, we should seek the right architecture to support it. This architecture must be sufficiently generic to encompass all the possible flows of information in the system and support the compositional approach described earlier but, at the same time, one with sufficient expressiveness to allow the identification of the several entities that make up the system, including the environment, the hardware and the software components.

We propose the **Generic-Events Architecture (GEAR)**, depicted in Figure 4, which we describe in what follows. The L-shaped structure is crucial to ensure some of the properties described.

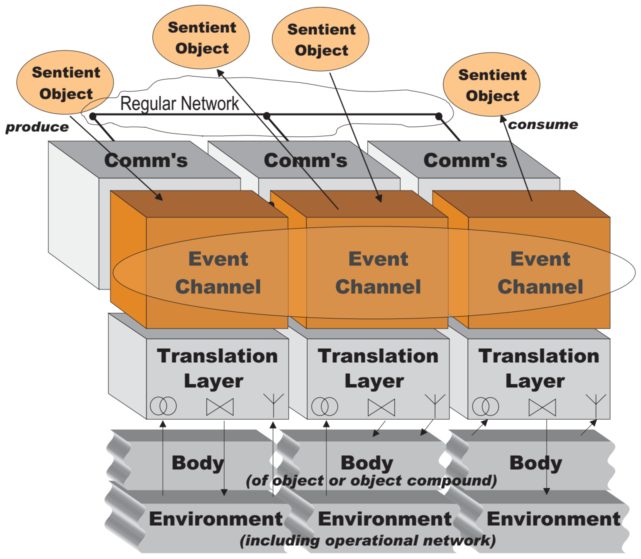

Figure 4. Generic-Events architecture.

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 (EC) dialect, between Environment/Body and Event Channel. Essentially one doing observation and actuation operations on the lower side, and doing transactions of event descriptions on the other.

Event Channel The layer responsible for event propagation in the whole system. 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 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 with 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.

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 *Generic-Events Architecture* (GEAR) introduces some innovative ideas in distributed systems architecture, which we discuss in what follows:

- It serves an object model based on production and consumption of generic events.
- Events are produced by several sources— environment, body, objects— which are all treated in a homogeneous way.
- There is a basic dialect for talking about events, used in all transactions by the Event Channel.
- The Translation Layer performs the transformation between the physical representation of a real-time entity and the EC compliant format, in either direction.
- The Event Channel propagates events through Regular Network infrastructures, via regular message-passing protocols.
- The flow of information (external environment and computational part) is seamlessly supported by the L-shaped architecture.

The GEAR architecture serves an object model based on production and consumption of generic events. A generic event in our model is a happening that requires the attention of objects which showed an interest on (*subscribed*) this class of happenings. An event is then *consumed* by the object when it happens. Events are presented to objects through an Event Channel, which is 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.

There is a basic dialect for talking about events, used in transactions by the Event Channel. Any entity speaking this dialect is capable of producing events to be propagated by the Event Channel to the relevant parts of the system, and of consuming events which are propagated by the EC. Some of these entities are sentient objects, which are capable of perceiving these events and perform processing on them. Other entities are simpler: smart transducers, sensors and actuators. They deserve the 'smart' adjective because they speak the EC dialect. However, they are limited in their ability: sensors can only generate EC compliant events which report what they sensed; actuators can only consume EC compliant events requesting an actuation, and perform the requested action.

The Translation Layer performs the transformation between the physical representation of a real-time entity and the EC compliant format, in either direction. Note that a special kind of device, represented in Figure 4 by an antenna, was introduced to denote that, unlike what happens in many models, it is necessary to englobe the operational network transmission and reception in the translation layer activity, and not in the regular network activity. By operational networking we mean network activity further to the regular packet passing network of a distributed system. This is very typical of real-time systems, namely those containing embedded components, and materializes in the form of I/O fieldbuses, IR beaming to control remote actuators, or radio beam reception from remote sensors.

The propagation mission assigned to the Event Channel is performed by Regular Network infrastructures (Ethernet, Internet, WLAN, RTLAN), through regular message-passing protocols. However, they must ensure whatever real-time, reliability and consistency properties are required of the relevant implementation [14]. The Communication Layer features protocols capable of enforcing the consistency properties of the information flow across the event channel platform, as suggested by the shade uniting local EC modules. The distributed EC is in fact a middleware platform, and its functionality may be completed by functions such as: routing, filtering, enforcement of delivery semantics. For example, filtering requires event channels to be identified by a subject, over which filtering can be performed. There exist approaches using content-based addressing over single channels [3], but this requires filtering every message. More appropriate in sentient environments is to use subject-based filtering, in which every event channel is identified by a particular subject. To improve performance, at the expenses of extensibility and anonymity, it is possible to bind event channels to the addressing mechanisms of the underlying network infrastructure.

The proposed architecture supports the flow of information 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. Body sensors produce events which report on the body. They are disseminated by the local EC module, through the Regular Network, to any relevant remote EC modules where entities showed an interest on them, normally, sentient objects attached to the respective local EC 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 the EC using the same EC dialect that serves, e.g., sensor 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 car computer system. 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: through the local EC 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 EC, to reach the former. In any case, if they are of interest to other entities, such as other sentient objects that wish to be informed of the actuation command, then they are disseminated on the EC to these sentient objects. A key advantage of this architecture is that messages and events can be globally ordered since they all pass through the event channel.

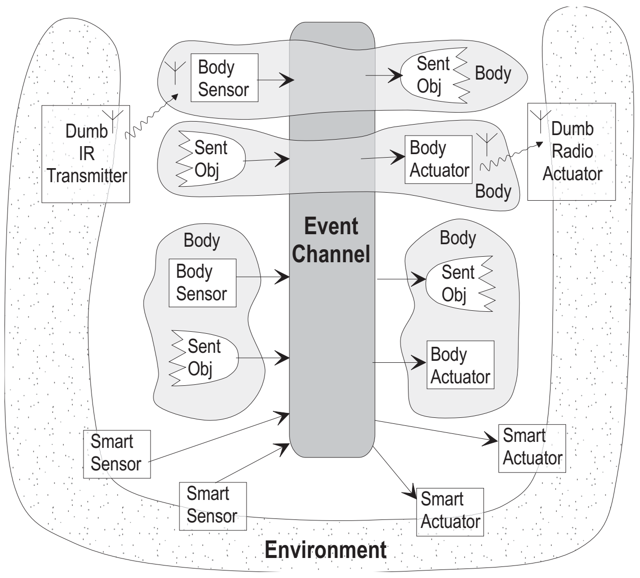

Figure 5. Information flow in the whole system.

One of the fundamental challenges we address in this paper is the provision of support for non-functional requirements. Our focus is on the aspect of timeliness. 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 can be done by establishing fundamental correctness criteria for the operation of the system. The system architecture, including the protocols and mechanisms lying in the middleware (which, in fact, can be part of the Event Channel), must be built so that the strict observation of the established criteria is ensured. Given the distributed nature of the problem, the correctness of the 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 Channel, where everything passes. As such, and very importantly, all consistency criteria that we define as to be secured by the EC, apply as well to regular messages, messages through other, operational network channels, and input/output feedback paths through the environment. In this architecture, no hidden channel problems affect the operation of the system [18].

Before proceeding, we need to define events more precisely. A **generic event** is a happening that takes place in the event channel at a given instant of the timeline, $\langle$ E,T $\rangle$. The happening is internal to the system, has an event-channel-compliant representation, and is not necessarily related with physical events taking place in the environment. That is, the 'event' is the happening as seen by the event channel, at a given instant of the timeline.

In the following section we present a small example to illustrate how an event channel aware of timeliness requirements can be used in a concrete scenario.

4 A Cooperating Cars Scenario

An interesting scenario to exercise some of the ideas presented in this paper, is one involving several sentient objects operating and interacting with their environment, and where the correct operation of the whole system requires some timeliness requirements to be met. A cooperating cars scenario provides all these ingredients [5].

Modern cars make an increasing use of sensor technology and advanced mechatronic systems. It is expected that in a near future they will be equipped with all kinds of sensors, for the diagnosis of internal components, to let them know their exact position, for the detection of obstacles, for the evaluation of weather and road conditions, etc. Based on all this information it will be possible to automate several controls and functionalities, specially all those not involving any safety issues. However, the automatic control of critical functions is a more challenging objective. The fundamental problem is that it is not possible to construct an exact 'image' of the surrounding environment, and therefore there is a risk of making wrong decisions based on incomplete or inaccurate information. Moreover, the environment is continuously changing, which introduces an additional time dimension that must also be taken into account. In fact, the intrinsic problem is that of *unpredictability*, which must be addressed in the context of adequate models and solutions.

In the scenario we consider, there may exist several cars (sentient objects) with the ability to 'consume' events from the environment and 'produce' events to it, trying to perform some function inside a restricted area, call it a zone, requiring the ability to perceive the environment and control their operation. We obviously model our system using the GEAR architecture, which implies that there exist Event Channels through which events are disseminated and received. Because there are several cars, the fundamental safety rule consists in ensuring that no car crashes occur (to focus our discussion, we do not consider other, static, obstacles). To satisfy this safety rule, each car must obviously be aware of each other car position. Therefore, we assume that each car periodically disseminates its position to the environment, which will be used by the interested cars (namely those lying in its proximity).

One possible way to achieve this would be by providing to all of them a temporally consistent image of the external environment, and one that would be consistent with each one's internal environment (what their body tells them), e.g., the coordinates and attitude of a car, and the state of internal variables. 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 the zone (the part of the environment circumscribed to a given zone). Now, suppose all sentient objects feature a module, call it constructor, in charge of building a *real-time image* of the zone. The constructor subscribes to the environment and body events, as well as to any relevant events produced by the other objects. 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 or not be made public. What is important is that *all* these events obey global consistency rules.

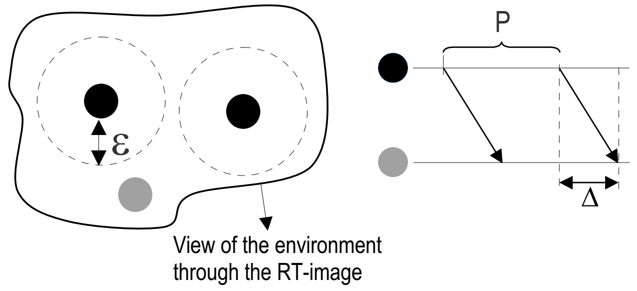

Figure 6. Safety rules in the cooperating cars scenario.

The safety rule is illustrated on the left of Figure 6. At every instant, each car must know the position of all other cars with a bounded error (ϵ). This error depends on how much time has passed since the position of a car was disseminated and on the maximum speed of that car. Therefore, both of them must be bounded. The control safety rule that must be satisfied is that the grey car cannot 'enter' inside the dashed circles.

The problem would be easily solvable if we could assume a reliable and timely propagation of events through EC modules. In such a traditional hard real-time approach, the system would be configured so that at every P time units the grey car would receive information from the black car (see Figure 6 on the right). Every car would be able to construct a real-time image of the environment (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. Then, every car would consistently decide what to do. However, because we are assuming a distributed environment, possibly with many cars disseminating events to a shared event channel, we have to consider the uncertainty of the environment. Hence, we also have to solve the conflict between this uncertainty and the requirements for timeliness (that is, ensuring bound Δ).

The problem of operating in uncertain environments is that timing bounds (e.g., like those resulting from temporal consistency constraints) may be violated because of timing failures. Therefore, when executing in uncertain environments, applications with timeliness requirements must be able to deal with timing failures. We assume our architecture, despite uncertainty of communication, is endowed with perfect timing failure detection, such as supported by a timely computing base (TCB) [17]. Fail-safe applications deal with timing failures simply by switching to a fail-safe state as soon as a timing failure occurs. In that way, they avoid any misbehavior and they preserve safety. This is only possible, however, if the failure is timely detected, before there is time to do anything wrong. The timely timing failure detection and timely execution of safety procedures can be done with the help of the TCB services.

Considering our cooperating cars scenario, it is quite easy to see that there exists a fail-safe state to which a car can switch if anything goes wrong: it just has to stop. Hence, it would be possible to construct an EC module with associated timeliness requirements (translated into timing bounds) and, simultaneously, use the adequate constructs to ensure that, would the real-time view of the environment at that car become temporally inconsistent, it would immediately stop. In this example, it is also clear that the car would be able to start moving again, as soon as its view of the environment would become consistent again.

Note that it would be possible to relax the timing bounds assumed for the propagation of events through the event channel, in order to reduce the probability of the occurrence of timing failures (and hence the probability of a car stopping). However, unless the position were disseminated more frequently, this would increase the error bound (ϵ) associated to the local image of the environment (which is also not good). To keep the error while increasing the assumed timing bounds, it would be necessary to reduce the allowed maximum speed. As a matter of fact, the possibility of adjusting the maximum car speed along with the assumed timing bound would be allowed for applications of the time-elastic class, that is, applications where bounds are automatically adjusted with the QoS of the supporting runtime environment [4].

But the relevant question has to do with knowing *when* and *how* to make the adjustments in order to have better results (i.e., for a car to keep moving, possibly slower, but never stopping because of a timing failure). Note that if the environment does not change, then it is not necessary to make any adjustments (assuming that the operational state is already the best). On the other hand, when the environment changes, the system should adapt to the new conditions. Once more, this problem may be solved by dependably estimating the state of the environment and deciding how to adapt. That is, by measuring the timeliness of the event channel, and making these measurements available to all EC modules: this would allow all relevant cars to adapt consistently.

5 Conclusion

We discussed an innovative model and architecture to support event-based object-oriented programming on real-time cooperative and embedded systems. The component-based approach to object definition is very important, as it reflects well the nature of the target systems. The body-environment duality is key to recursive composition in the model. The L-shaped block diagram allows the harmonious confluence of physical and computerized information flows. To our knowledge, it is the first architecture to provide 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 embedded systems, which we intend to explore in future papers. The generic event concept provides a common representation inside the system for what used to be "state" and "event" messages, and this opens an avenue for the integration of event-triggered and time-triggered operation, a subject of current importance in embedded systems.

References

[1] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, aug 2001.

[2] 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.

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

[4] A. Casimiro and P. Veríssimo. Using the timely computing base for dependable qos adaptation. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, pages 208–217, New Orleans, USA, Oct. 2001. IEEE Computer Society Press.

[5] Definition of application scenarios. CORTEX project, IST-2000-26031, Deliverable D1, Oct. 2001.

[6] Preliminary definition of cortex programming model. CORTEX project, IST-2000-26031, Deliverable D2, Mar. 2002.

[7] I. Crnkovic and M. Larsson, editors. *Building Reliable Component-Based Software Systems.* Artech House Publishers, 2002.

[8] O. M. Group. The common object request broker: Architecture and specification. Technical Report OMG Document 96-03-04, July 1995.

[9] 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.

[10] A. J. Herbert, J. Monk, and R. van der Linden. *The ANSA Reference Manual, Release 1.1.* Architecture Projects Management, Ltd, Cambridge, UK, July 1989.

[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] M. Horstmann and M. Kirtland. Dcom architecture. http://www.microsoft.com/jini/specs/.

[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] 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.

[16] 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.

[17] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *Transaction on Computers - Special Issue on Asynchronous Real-Time Systems*, 51(8), Aug. 2002. A preliminary version of this document appeared as Technical Report DI/FCUL TR 99-2, Department of Computer Science, University of Lisboa, Apr 1999.

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