The CORTEX Programming Model

G. Biegel, G. Blair, C. Brudna, V. Cahill, A. Casimiro,
S. Clarke, H. Duran-Limon, A. Fitzpatrick, A. Friday,
B. Hughes, J. Kaiser, R. Meier, V. Reynolds,
P. Veríssimo and M. Wu

DI-FCUL TR–03–19

July 2003

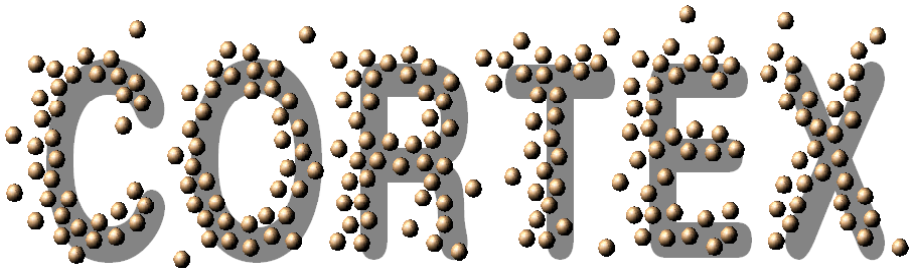
Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1700 Lisboa
Portugal

Technical reports are available at http://www.di.fc.ul.pt/tech-reports. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Project IST-2000-26031

CO-operating Real-time sentient objects:

Architecture and Experimental evaluation



The CORTEX Programming Model

CORTEX deliverable D6

Version 1.0

April 24, 2003

Revisions

Rev.	Date	Comment
0.1	17/4/03	First draft circulated to partners
1.0	24/4/03	Final Version produced and submitted

Editor

Vinny Reynolds, Trinity College Dublin

Contributors

Greg Biegel, Trinity College Dublin
Gordon Blair, University of Lancaster
Cristiano Brudna, University of Ulm
Vinny Cahill, Trinity College Dublin
António Casimiro Costa, University of Lisbon
Siobhán Clarke, Trinity College Dublin
Hector Duran-Limon, University of Lancaster
Adrian Fitzpatrick, Trinity College Dublin
Adrian Friday, University of Lancaster
Barbara Hughes, Trinity College Dublin
Jörg Kaiser, University of Ulm
Rene Meier, Trinity College Dublin
Vinny Reynolds, Trinity College Dublin
Paulo Verissimo, University of Lisbon
Maomao Wu, University of Lancaster

Address

Dept of Computer Science,
Trinity College Dublin,
Ireland

Table Of Contents

Executive Summary

This deliverable describes work that has been carried out during the second year of the project for work package WP1. The objective of the work package as defined in the technical annex, is 'the design of a programming model that supports the development of *proactive* applications constructed from mobile *sentient objects'*. The work is divided into four tasks, with Task 1.1, application requirements**,** having been completed as deliverable D1 in month 6 of the project. This deliverable addresses the three remaining tasks. Task 1.2 deals with the definition of a *sentient object model* in a language independent way. Task 1.3 defines an event based communication model for sentient objects including mechanisms for controlling the propagation of events in the system based on both physical proximity and event content. Task 1.4 defines mechanisms for the specification of Quality of Service (QoS) parameters that may be mapped to the system level.

The deliverable starts by addressing Task 1.2 and firstly provides a definition of the sentient object model. Concrete definitions of the primitives, *sensor, actuator* and *sentient object* are provided and the internal structure of a sentient object with regard to sensory capture, context-awareness and intelligent inference is examined. This chapter continues with an examination of how the sentient object model deals with the uncertainties inherent in information sensed from the physical environment. A probabilistic sensor fusion scheme based on Bayesian networks is proposed as an approach to managing uncertainty in context-aware applications based on the sentient object model. The chapter concludes with an examination of machine learning techniques that may be incorporated into the sentient object model to provide service adaptation according to interests, preferences, knowledge or goals.

Chapter 3 of the deliverable addresses Task 1.3 and describes the event based communication model used by sentient objects. The chapter begins by describing the development of an event model designed to address the predictability requirements of applications operating in mobile environments based on the CORTEX WAN-of-CANs network. This event-based programming model incorporates the topology of a heterogeneous communication infrastructure as specified in CORTEX and an abstract network model, reflecting the different levels of quality of service available from the different sub networks, is provided. The programming model is then based on the concept of event channels with different temporal and reliability guarantees that may be mapped to certain QoS zones. This chapter goes on to describe a conceptual model designed to alleviate the impediments to real-time event-based communication characteristic of wireless, mobile ad hoc environments as found in CORTEX applications. The model is based on a predictive architecture combining mobility prediction with partition anticipation to achieve predictive routing and resource reservation and thus limits the unpredictability of wireless communication.

Finally, Chapter 4 of the deliverable addresses Task 1.4, that of Quality of Service (QoS) specification. The model for timeliness QoS requirements specification based on the specification of <bound, coverage> pairs is extended to a more general way of specifying QoS requirements not limited to the observation of time bounds. The extended approach permits for the integration of other application requirements into the QoS framework and provides a more flexible and configurable approach to the specification of QoS requirements.

A description language for the specification of both the task model and the resource model of sentient objects is presented as an extension of the resource configuration description language (RCDL). This language provides the facility for programming of resource

requirements for sentient objects and is part of the QoS specification aspect of the CORTEX programming model.

Chapter 1: Overview

The objective of D6 is to design a programming model suitable for the development of proactive applications constructed from mobile sentient objects. D6 embodies the final deliverable and follows its predecessor deliverable D2, the preliminary definition of the CORTEX programming model.

The programming model should support all aspects of the behaviour of sentient objects. This includes acquiring information from the environment, being context aware in an uncertain situation, reacting in an appropriate manner to that situation and thus modifying the state of the environment. In the inherently mobile and dynamic WAN of CAN structure, the programming model must also provide an event based communication model and associated Quality of Service and timeliness guarantees.

D6, The Programming Model is presented as a collection of technical reports and an API, some of which have been submitted fro publication. The deliverable is divided into three major parts, each of which contains one or more technical reports or papers discussing the corresponding objective.

1.1 Chapter Outline

Chapter 2 outlines our definition of the context aware sentient objects and specifies the internal components of such an object. Chapter 2 then describes how an application developer may program the sentient objects in a context aware manner using techniques such as Bayesian networks to deal with uncertainty and Machine Learning to improve performance by learning from past experiences

In Chapter 3, D6 describes the CORTEX event-based communication model made available to the application programmer for defining inter-object communication. In particular, an event model is described which addresses the predictability requirements of applications operating in mobile environments. Furthermore, Chapter 3 describes the impediments imposed by ad hoc wireless networks on real time event based communication and then proposes a high level model to reduce their impact.

Chapter 4 presents our approach to specifying QoS parameters characterizing the level of service supported by the underlying, heterogeneous infrastructure and for reserving the resources required to enforce the specified QoS. Chapter 4 also provides a description language which gives the specification of the task model and the resource model of sentient objects by expressing the resource requirements of their components

In the Appendix, an API is presented for the Event Channel classes. Three Event Channel classes are presented in the API: Hard Real Time, Soft Real Time and Non Real Time Event channels

Chapter 2: Sentient Object Model

2.1 Towards a Sentient Object Model

Adrian Fitzpatrick, Gregory Biegel, Siobhan Clarke, Vinny Cahill

Distributed Systems Group

Department of Computer Science

Trinity College

Dublin 2, Ireland

{firstname.lastname}@cs.tcd.ie

Abstract—A sentient object is a mobile, intelligent software component that is able to sense its environment via sensors and react to sensed information via actuators. Sentient objects are context-aware, aware of both their internal state and the state of their surrounding local environment, and are cooperative, cooperating with other sentient objects both through traditional communication channels and via the physical environment. In this paper we describe a sentient object model and in doing so provide concrete definitions for the terms sensor, actuator, and sentient object as used in our model.

Keywords— Sentient computing, mobile computing, context-aware, stigmergy

2.1.1 Introduction

The continued evolution of computing and communication technologies towards ever smaller and more powerful devices has led to a new generation of applications where computing power is widely deployed throughout the environment. The development of such pervasive computing environments has also been driven by the availability of improved sensor technology providing accurate and trustworthy sensing at affordable prices. Cheap, ubiquitous sensors coupled with improved computing power and wireless data communications have made a new class of decentralised and proactive applications possible. It is envisaged that this class of applications will consist of a very large number of mobile software components accepting input from the environment via a variety of sensors and autonomously acting upon the environment via a variety of actuators. These components will contain intelligence to allow them to act autonomously based upon acquisition of information from the environment, and will cooperate with each other using a range of different networking technologies. It is these mobile, intelligent software components that we term sentient objects and in this paper we present a model for the development of such objects. We provide concrete definitions for the terms sensor, actuator and sentient object. Context awareness [1] is a characteristic of reactive sentient objects and this is dealt with in our model through the use of contextual information gleaned from sensors to control the actions of a sentient object. Finally, our model addresses the coordination of multiple objects, via the physical environment using biologically inspired mechanisms.

The possible applications of sentient objects are numerous and diverse and include such areas as intelligent vehicles, smart buildings and mobile robotics.

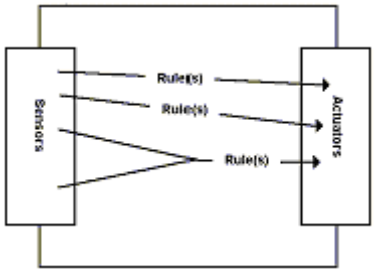


Figure 1. Simple sentient object model

2.1.2 Related Research

[2] defines a sentient computing system as one where an application appears to share the user's perception of the environment. They have developed a sentient system that uses wireless radio transceiver sensors to maintain a software model of the location of a set of laboratory users and objects. This model is then used for a number of intelligent 'follow-me' applications. Similarly, López de Ipina defines sentient systems as systems that respond to stimuli provided by sensors distributed throughout the environment by triggering actions that are adequate to the changing context of the user [3]. His TRIP system uses location, identification and orientation information to provide context aware services to users. The Sentient Information Framework, part of the TRIP system is a programming framework designed to separate context capture and abstraction from application semantics and provide efficient mechanisms for context communication [4]. Context Based Reasoning is a paradigm introduced by Gonzalez [5] as a concise but rich representation paradigm that could be used to model the intelligent behaviour of opponents in simulations. The hypothesis behind CxBR is that the actions taken by an intelligent entity are highly dependent on the entity's current situation (context). Following this approach and limiting the number of actions permitted in a certain context, the efficiency of rule-based inference may be increased substantially.

2.1.3 A Sentient Object Model

Essentially a sentient object is an encapsulated entity, with its interfaces being sensors and actuators. The actuators are controlled according to sensor input, perhaps via a rule based inference engine, as depicted in Figure 1.This simple view does not, however, answer many questions about sentient objects. For example,

- What do we mean by sensors and actuators in this context? Are they hardware devices, or software abstractions thereof?
- What is the granularity of a sentient object?
- How do sentient objects interact?
- What sorts of hierarchy/relationships can exist between

In addition to answering the questions above, one of the main challenges to be overcome in defining a sentient object model was to constrain the definition. The simple definition above can easily encompass a great many existing computer systems. For example, a desktop computer could be said to sense user input via the mouse or keyboard and actuate on its environment by moving a cursor or displaying a character on the monitor. To be useful, a

more precise definition of a sentient object suited to the pervasive computing applications of interest is need.

A. *Event based communication*

We expect sentient objects to interact using an anonymous, event-based inter-process communication paradigm that supports loose coupling between sentient objects in order to provide for object mobility and application evolution. We identify two distinct categories of events: software events and real-world events. Software events are the main form of interaction between entities in our model and provide anonymous, ad-hoc communication. real-world events are anything that happens in the environment, either causing a change of state in a sensor or caused by an actuator. We propose that, based on these two categories of events, we can identity and distinguish the three major entities that may exist in our sentient object model.

B. *Classifications*

The different possibilities for production and consumption of events lead to the different classifications. Our initial investigation of the different possibilities for production and consumption considered that an entity could produce only one category of event and consume only one category of event. The resulting classification yields four distinct entity types, through which we aim to provide a separation of the semantics of information acquisition, application logic and environmental actuation.

• *Real-world consumption, software production*

This class of entity produces software events in response to real world events consumed. In effect, it provides information about a physical occurrence in the external environment by translating the information into the format of a software event, and by releasing that event into the event-based middleware environment. This is classical sensor functionality; therefore all entities in our model that conform to this classification will be termed sensors.

• *Software consumption, software production*

This class of entity both consumes and produces software events, implying the flow of information to and from these entities is purely through the event-based middleware. It is in this middleware domain that the application logic will reside, therefore these entities will provide the logic building blocks for CORTEX applications. Entities fitting this classification will be called sentient objects.

•*Software consumption, real world production*

These entities produce real world events in response to the consumption of software events. They have the opposite consumption/production properties to the entities that we have identified as sensors, and hence we designate these entities actuators

•*Real-world consumption, real-world production*

This class of entity produces real-world events in response to real-world events consumed. This implies that it is a simply real-world system; in the context of our model, we propose to term such entities sentient systems.

It is from these classifications according to event consumption and production that we derive the definitions for the three major entities in our sentient object model.

B.1 *Sensor*

A sensor is defined in [6] as being a device that responds to a physical stimulus, such as thermal energy, electromagnetic energy, acoustic energy, pressure, magnetism or motion, by producing a signal, usually electrical. We refine this traditional definition of a sensor and in our sentient object model define a sensor as

An entity that produces software events in reaction to a real-world stimulus detected by some world hardware device

Our definition of a sensor is illustrated in Figure 2.

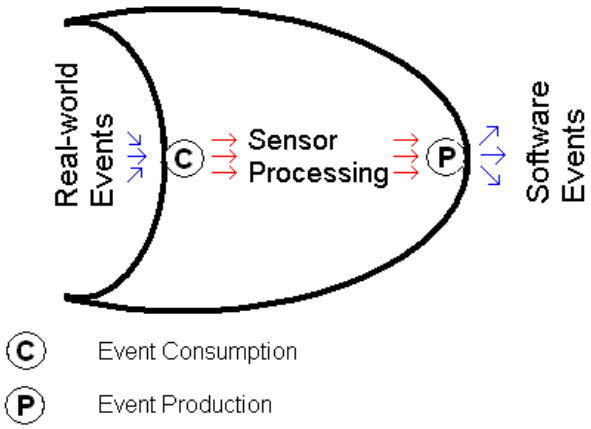


Fig. 2. A sensor

The *sensor processing* task encapsulated in our definition of a sensor refers to the possible abstraction of raw sensor data into more useful information by the sensor itself, before the production of software events. An example of such abstraction may be the transformation of raw GPS coordinates into more useful location information e.g. the conversion of coordinates 53 23' N, 6 20'Wto the location "Room G15, O'Reilly Institute, Dublin, Ireland". Such sensor processing is not necessarily carried out by a sensor but may potentially be.

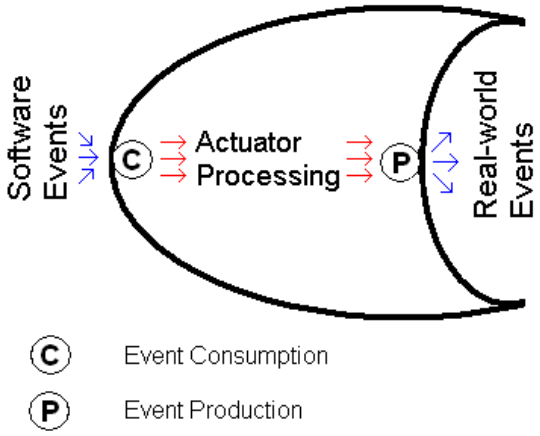


Fig. 3. An actuator

B.2 *Actuator*

[7] defines an actuator as a mechanical device for moving or controlling something. Our definition of an actuator in the sentient object model maintains that it is something that causes a change in the physical environment, but narrows the definition in terms of what causes the actuation to occur. We define an *actuator* in our sentient object model as

an entity that consumes software events, and reacts by attempting to change the state of the real world in some way via some hardware device

Our definition of an actuator is illustrated in Figure 3, where the *actuator processing* task refers to potential transformation of incoming software events before the production of hardware events.

B.3 *Sentient Object*

Following our definitions of sensor and actuator, we define *a sentient object* as

an entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator

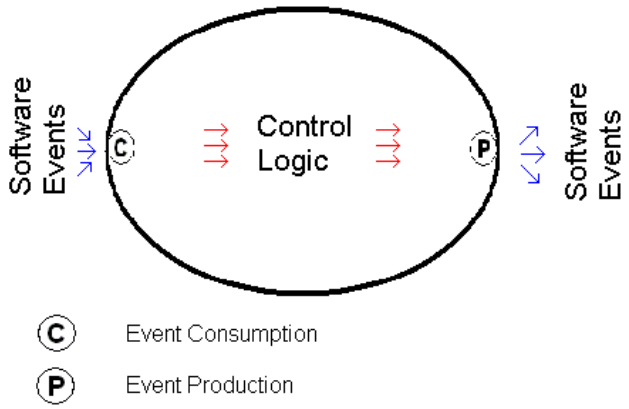


Fig. 4. A sentient object

We have previously stated that sentient objects will be the building blocks of logic for sentient object applications. Aside from our identification of sentient objects through the classification of events consumed and produced, we introduce into our definition the additional stipulation that a sentient object should exist in a control path between a sensor and an actuator. We do this in order to constrain our definition somewhat, as we feel that to provide a definition that simply specifies the consumption and production of software events is too general, and that many currently existing software entities fall into that classification.

Up until this point, we have not discussed what form the *control logic* in the sentient object takes. The internal control logic must exploit context-awareness and is discussed in the next section.

2.1.4 Context Awareness in Sentient Objects

Essentially sentient objects sense and interact with their environment via sensors and actuators. It is this awareness of, and interaction with the environment that makes context awareness an important factor in sentient objects. Before examining the role of context in our sentient object model, a clear definition of what we understand by context is required. There are multiple definitions of *context* available in the literature [8], [1], [9].

For the purposes of the sentient object model, we propose a definition of context as

Any information sensed from the environment that may be used to describe the situation of a sentient object. This includes information about the underlying infrastructure available to the sentient object.

Our definition of *context-aware* then follows,

The use of context to provide information, to a sentient object, which may be used in its interactions with other sentient objects, and/or the fulfillment of its goals.

We have identified three components necessary for context-awareness in a sentient object.

A. *Context Acquisition*

A sentient object may receive input from an array of diverse sensors, for example a sentient vehicle's array of sensors could include proximity sensors, GPS, speed and direction sensors, and pollution sensors. Signals from these sensors need to be integrated in order to determine the overall environment and context of the sentient object. In addition to the problem of fusing data from such diverse sensors, each data source has an error associated with it. The major issues to be addressed in the area of sensory capture are data filtering and sensor fusion

B. *Context Representation*

Raw sensor data will usually need to be transformed in some way before it may be considered useful contextual information. Such transformation may occur in the sensor itself as discussed in Section III-B.1, or may be carried out within the sentient object itself. The context representation component deals with the representation of context information in a way that is useful to the sentient object and may be easily exchanged amongst sentient objects. We are currently examining the use of XML in context representation for sentient objects.

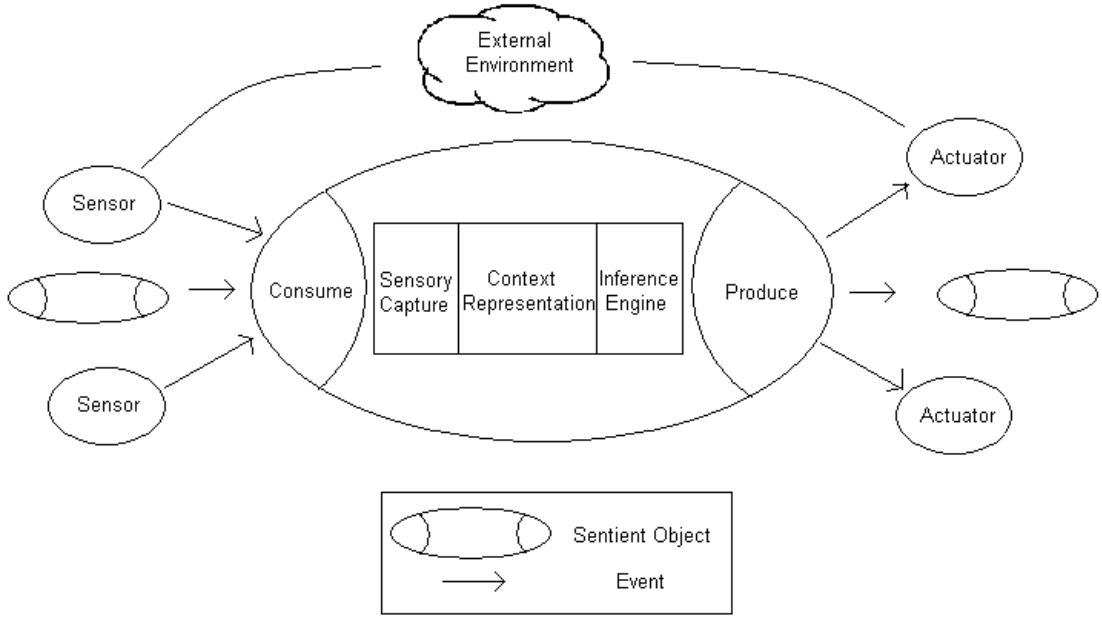


Fig. 5. A sentient object showing the components of the internal control logic

C. *Inference*

Sentient objects are expected to act upon their environment, changing its state. This implies some form of decision making ability or intelligence, on the part of the sentient object that is captured in the inference engine component. *An inference engine*, in artificial intelligence terms, refers to a program that reasons about a set of rules (a knowledge base) in order to derive an output.

The knowledge base of an inference engine contains the knowledge required to solve a certain problem, encoded as a set of *production rules*. The knowledge encoded in such rules is generally captured from a human expert who is able to express his expertise in the form of such rules.

The inference engine should be as generic as possible so that it may be applied to a number of different knowledge bases in different domains with minimal changes to itself. For the purposes of sentient objects, we can envisage a sentient object inference engine, with object and domain specific knowledge bases in each object.

We are currently focusing on using rules specified in CLIPS (C Language Integrated Production System) [10], a declarative language with a built in inference engine based upon

the RETE net [11]. We incorporate the mechanism of Context Based Reasoning (CxBR) [5] to limit the number of production rules to be considered and to increase the efficiency of the inference process.

2.1.5 Coordination through the environment

Coordination of actions through the environment or *stigmergy*, was first observed in colonies of insects [12] cited in [13], and describes coordination of the activities of individuals without direct communication between them. Stigmergy is a highly decentralised method of coordination where individual entities follow the same set of simple behavioural rules to yield robust and adaptive coordination, without the need for expensive and unreliable communication.

In stigmergy, coordinated behaviour arises from individuals sensing their physical environment and reacting to the sensed information according to a simple set of rules. The stimulus may come from the physical environment itself, in a type of stigmergy known as *sematectonic* stigmergy, or may come from something that makes no direct contribution to the task at hand and is used solely to influence behaviour in a form of stigmergy known as *sign-based* stigmergy.

We propose stigmergy as a coordination mechanism for very large networks of sentient objects. Sentient objects contain behaviours encoded as rules in the inference engine and achieve coordinated behaviour through sampling and acting upon their environment.

2.1.6 Conclusions

We have presented a sentient object model that we hope will contribute towards the development of context-aware applications. As part of our model we provide definitions for the terms sensor, actuator and sentient object and describe the internal structure that enables sentient objects to operate autonomously.

References

[1] Anind K. Dey and Gregory D. Abowd, Towards a Better Understanding of context and context-awareness, Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, June 1999

[2] Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward and Andy Hopper, Implementing a Sentient Computing System, IEEE Computer, Vol.34, No. 8, Aug 2001.

[3] Diego Lopez de Ipina, An ECA Rule-Matching Service for Simpler Development of Reactive Applications, In Proceedings of Middleware 2001, IEEE Distributed Systems Online, Vol 2, No. 7, November 2001

[4] Diego Lopez de Ipina, Building Components for a Distributed Sentient Framework with Python and CORBA, Proceedings of the 8th International Python Conference, Arlington, VA, USA. 24-27 January, 2000

[5] Avelino J. Gonzalez, Robert H. Ahlers, Context-based Representation of Intelligent Behavior in Training Simulations, Naval Air Warfare Center Training Systems Division, 1998

[6] United States Government, Federal Standard 1037C - Glossary of Telecommunications Terms, August 7, 1996

[7] Merriam-Webster, Inc. Merriam-Webster Collegiate Dictionary, 10th Edition, 1998

[8] Schilit, B., Adams, N. Want, R. Context-Aware Computing Applications 1st International Workshop on Mobile Computing Systems and Applications. (1994) 85-90

[9] Guanling Chen and David Kotz A Survey of Context-Aware Mobile Computing Research, Department of Computer Science, Dartmouth College Technical Report TR2000-381, November 2000.

[10] NASA CLIPS: A Tool for Building Expert Systems

http://www.ghg.net/clips/CLIPS.html

[11] Forgy, C.L Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem Artificial Intelligence 19, 1982

[12] Grasse, P.-P., Le reconstruction du nid et les coordinations interindividuelles chez Bellicositermes Natalensis et Cubitermes sp.. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs, Insectes Sociaux, vol. 6, 41-81 1959.

[13] Marco Dorigo, Eric Bonabeau, Guy Theraulaz Ant algorithms and stigmergy Future Generation Computer Systems 16 (2000) 851-871.

2.2 Towards Middleware for Coping with Uncertainty in Context-Aware Applications

Gregory Biegel and Vinny Cahill

Distributed Systems Group

Trinity College Dublin

{Greg.Biegel, Vinny.Cahill}@cs.tcd.ie

Abstract

Uncertainty is a major problem in sensing the environment due to the inherent limitations of sensors with respect to accuracy and precision. This has led to a crucial requirement for middleware that provides uncertainty management for software components whose actions are based on environmental perception. Sentient objects are context-aware, mobile, intelligent software components able to sense the environment using a variety of sensors and make changes to the environment by way of actuators, independently of human control. Sentient objects rely on the fusion of outputs from a variety of sensors in order to determine their context and perform actuation using the Context-Based Reasoning paradigm. In this paper we propose a middleware architecture based on the sentient object model and incorporating a probabilistic sensor fusion scheme combining Bayesian networks and Context-Based Reasoning, for coping with uncertainty in context-aware applications.

2.2.1 Introduction

As applications continue to move away from the desktop and into the physical world, the development of a new class of *decentralised* and *proactive* application that makes use of computing power widely deployed throughout the environment has become possible. The development of such *pervasive* computing applications has been driven by the continuing evolution of computing and communication technologies coupled with the availability of a range of cheap and diverse sensing technologies, enabling measurement of diverse aspects of the environment. We envisage this class of application as consisting of a very large number of mobile software components accepting input from the physical environment via a variety of sensors and autonomously acting on the environment via a variety of actuators. Environmental information gleaned from sensors allows these components to be *context-aware*, that is to detect and make use of changing environmental conditions to influence further actions. It is these software components that we term *sentient objects* and one of the major problems faced in the development of such objects is how to deal with the uncertainty associated with measurements of the physical environment made by sensors.

Sentient objects have a number of characteristics that are important in pervasive computing environments

- *Sentience* - the ability to perceive the state of the environment via sensors
- *Autonomy* - the ability to operate independently of human control in a decentralised manner
- *Proactiveness* - the ability to act in anticipation of future goals or problems

In this paper we describe the development of middleware, based on the sentient object model [1], which serves to insulate applications from the complexities of physical sensing technologies and the fusion of multi modal sensor data. It is our aim to make context acquisition and the uncertainty associated with this task, as transparent as possible to the application programmer.

Because the information we receive from sensors is incomplete, uncertain and contains errors we cannot rely on the output of a single sensor. In order to obtain sufficient correct information from the environment to reliably determine the context of a sentient object, we need to *fuse* the output of multiple disparate sensors. Although a significant body of work exists in the area of context-aware computing, we feel the important area of sensor fusion for dealing with uncertainty has not been adequately addressed in a way that may be applied to a range of applications. Most sensor fusion systems described in the literature are application specific and only address a specific set of sensors applied to a specific task [2, 3]. We propose a generic means to develop context-aware applications based on the sentient object model which incorporates Context-Based Reasoning (CxBR) [4], enhanced with a probabilistic sensor fusion scheme, to achieve context-aware behaviour.

A probabilistic approach to sensor fusion allows quantification of uncertainty or data reliability and is grounded in well-understood probability theory. Probabilistic models are attractive as they allow formal definition of the likelihood and level of belief in conclusions drawn from uncertain data. The derivation of context from multiple sensors is difficult and prone to error, but probabilistic models provide us with the means to measure the effectiveness of our derivations. We base our sensor fusion system for sentient objects on Bayesian networks [5], which are a class of probabilistic model that encodes conditional independence relationships between a set of random variables. The CxBR paradigm is enhanced with the addition of Bayesian networks as a means of fusing sensor data and managing uncertainty.

2.2.2 Related Work

The Context Toolkit [6] developed at the Georgia Institute of Technology provides a generic approach to developing context-aware applications and addresses the separation of context acquisition from the use of context information by applications through introduction of the context widget, a sensor abstraction which hides the underlying details of the sensing mechanism from the application. The Context Toolkit does not provide a mechanism for dealing for the uncertainty of sensor data, nor does it address the fusion of multiple sensor outputs in order to reduce such uncertainty. Applications subscribe to all pieces of context information in which they are interested and it is the responsibility of the application to perform any fusion of this information. The TEA project [7] aims to provide a similar approach to context-awareness with the abstraction of physical sensors by way of cues, the outputs of which are used to determine which context an object is in. The system makes use of semantic nets to represent the set of possible contexts and each context defines what may happen when entering, leaving or being in that context in a similar manner to Context-Based Reasoning [4].

The Multimedia Systems Laboratory at UCLA has developed the Multi-Use Sensor Environment (MUSE), which is a middleware architecture for sensor-rich smart spaces and which employs Bayesian networks for sensor fusion [8], whilst Microsoft has incorporated Bayesian networks into a number of its products, including the Office assistant [9] and the spam filter in Outlook [10].

2.2.3 The Sentient Object Model

A sentient object is an encapsulated entity, with its interfaces being *sensors* and *actuators*. Actuators are controlled according to contextual information gleaned from sensor inputs via an intelligent production rule-based inference engine. A sentient object and its internals are illustrated in Figure 1.

Sensors in the sentient object model are defined as entities which produce software events in reaction to a stimulus detected by some real-world hardware device. In this way, sensors in the sentient object model are a software abstraction or type of virtual sensor, providing an estimate of some environmental variable of interest which may in turn be used to derive the overall context of the object. The term *sensor* thus encapsulates a physical device, as well as software, which potentially provides a higher level symbolic interpretation of the output of the physical transducer.

An actuator in the sentient object model is defined as an entity that consumes software events and reacts by attempting to change the state of the real world in some way via some hardware device. Actuators in the sentient object model are software abstractions of actual physical devices, which consume events.

A sentient object is then defined as an entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator. Sentient objects are *cooperative* and communicate with each other and with sensors and actuators via an anonymous generative event based communication paradigm [11], permitting loose coupling between objects which supports object mobility and application evolution.

2.2.3.1 Context-Awareness

Humans use an implicit understanding of their environment and "context" in order to readily make complicated inferences and decisions. Such awareness is composed of many levels of knowledge including information about actions and intentions, as well as knowledge of the state of the environment. In the human domain it is clear that an accurate representation of context information depends on a diverse set of sensors, each measuring different targets with different resolutions and accuracies.

In terms of sentient objects, *context* is defined as any information sensed from the environment that may be used to describe the situation of a sentient object. This includes information about the underlying infrastructure available to the sentient object. Context-awareness is then the use of contextual information by the sentient object in fulfillment of its goals.

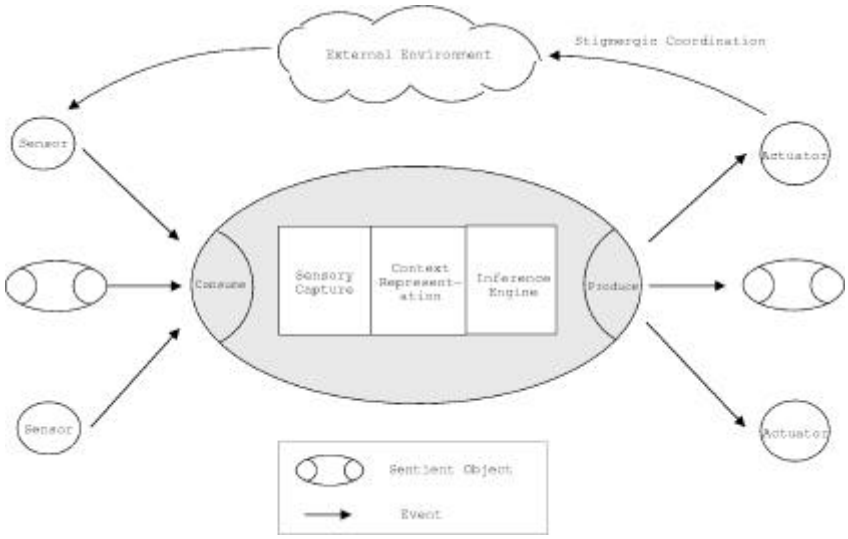


Figure 1: Sentient Object Model

Whilst the problem of fusing the output of multi modal sensors for the purposes of determining context has been solved in humans, this cannot be said for context-aware computing applications where no generalised solution exists for the mapping of sensor output to context representation. The way in which humans deal with large amounts of contextual information is by doing so in a hierarchical manner and sentient objects follow a similar model known as Context-Based Reasoning [4].

2.2.3.2 Context-Based Reasoning (CxBR)

The sentient object model embodies the concept of Context-Based Reasoning (CxBR). The paradigm derives its name from the idea that the actions taken by an entity are highly dependent on the entity's current context [4], that is a recognized situation defines a set of appropriate actions and the identification of a future situation is simplified if all possible actions are limited by the current situation itself. Context-Based Reasoning is based on the following hypotheses:

1. Small, but important portions of all available environmental inputs are used to recognise and treat the key features of a situation
2. There are a limited number of things that can realistically take place in any situation
3. The presence of a new situation will generally require alteration of the present course of action to some degree

CxBR encapsulates knowledge about actions to be taken and possible future situations into *contexts*. By defining a hierarchy of contexts in which a sentient object may exist and by associating specific actions to be undertaken in each context, a sentient object's behaviour may be influenced by its context. Following this approach and limiting the number of actions permitted in specific contexts, the efficiency of production rule-based inference may be increased substantially.

CxBR defines *a mission context, major contexts* and *sub contexts*. The mission context is the overall goal and objectives for a certain scenario. It is composed of major contexts, which are tactical operations assisting in the achievement of the scenario. Each major context is in turn composed of one or more sub contexts each of which is a lower level tactical procedure which assists in the achievement of its associated major context. Autonomous behaviour in a CxBR-

based entity is based upon the evaluation of a set of rules, and the alteration of the course of behaviour based on the outcome of these rules. The influence of behaviour based upon a set of continuously evaluated rules is a common Artificial Intelligence technique, but CxBR adds to this the notion of an *active context*. Within each active context (be it mission, major or sub), only a subset of the rules is applicable (this derives from the hypothesis that there are only a limited number of things that can realistically take place in any situation) and this increases the efficiency of a rules-based approach. Contexts are constantly being activated and deactivated during the entities lifetime and each context regulates the behaviour of the entity and provides an expectation for the future. A requirement of this approach is that certain cues exist to indicate when transitions may be made between active contexts.

An example CxBR hierarchy for a sentient vehicle is illustrated in Figure 2. In this example it is clear that in the **two-lane highway** mission context, whilst the **changing lane** major context is active, the possible sub contexts that the vehicle may be in are **accelerate** and **decelerate**, each of which has different actions associated with it.

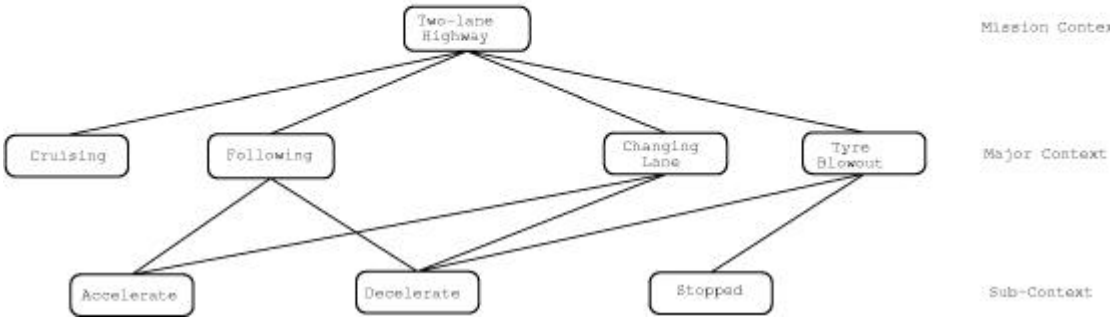


Figure 2: A CxBR hierarchy for a sentient vehicle

2.2.3.3 Sentient Object Internals

The internals of a sentient object as illustrated in Figure 1 consist of three major components. Each object has a sensory capture component that collects aspects of an objects context from a set of sensors. Such context data may include location, activity, proximity and infrastructural information. It is the sensory capture component that is responsible for the fusion of multi modal sensor input prior to the mapping to the context representation.

The *context representation* component consists of an object specific context information model [12] as well as a CxBR-type hierarchy of contexts in which the object may exist. Representation of context information in a way which is useful to a sentient object, or indeed any computing application, is a difficult problem which has not been extensively addressed in the literature. Albrecht Schmidt et al. [13] propose a hierarchically organised feature space for context classified broadly into human and environmental factors. For autonomous, software components like sentient objects, human factors are not an important consideration in context derivation, whereas environmental factors are more so. Anind Dey et al. classify four types of context that they see as being more useful practically, these being *location, identity, activity* and *time* [6]. In the sentient object model we treat sensor inputs as providing environmental information that includes location, proximity, visual, infrastructural and temporal information. These inputs are then used to determine the current context and consequently the position in the CxBR hierarchy. Each object has a specific context information model which specifies

what context information is important to the object. This context information model is composed of an XML-schema, where context information is represented symbolically as numbers and strings.

The inference engine component is a production-rule based inference engine and supporting knowledge base, which gives sentient objects the ability to intelligently control their actions based on their context. The inference engine is related to the CxBR hierarchy in that only a subset of the rules are valid in a given context which follows from the hypotheses that there are a limited number of things that can happen in any scenario.

2.2.4 Context Uncertainty Management

Uncertainty is an intrinsic factor in all sensing processes. In order to reason effectively in terms of context, we need a means for specifying and managing the uncertainty inherent in that context derivation.

2.2.4.1 Uncertainty of Sensor Data

Sensors that sense aspects of the physical environment, such as temperature, audio and location sensors provide as output only approximations to the values, which they sense in the real world. This uncertainty regarding the true value of what is being measured is inherent in data resulting from a physical measurement and results from hardware limitations in the manufacturing of the sensor and the fact that the physical operation of the sensor is too complex to model.

In addition to errors in physical sensor measurements, uncertainty in sensor data may also arise from the inability of a single sensor to sense all aspects of the environment of interest. A human example is stereovision in which the brain fuses the output of two eyes in order to provide depth perception, which is not available from the output of a single eye. In this way sensor fusion provides additional information about the environment and thus reduces uncertainty.

2.2.4.2 Sensor Fusion

When dealing with sentient objects, we are concerned with obtaining the most complete representation of the current context and indeed the most accurate mapping of the actual context to the active context in the CxBR hierarchy. By fusing the output of multiple sensors measuring both the same and different parameters of the environment at a single point in time, we may increase the probability of accurately identifying the context. Arnoud Visser [14] defines three general types of sensor fusion

- *Complementary fusion*: refers to the fusion of the output of multiple disparate sensors each giving partial information about the environment, and resolves incompleteness of data.
- *Competitive fusion*: this is the fusion of uncertain data from a number of sensors and aims to reduce the uncertainty of an individual sensor reading. A number of sensors measure the same parameter and multiple measurements are fused to provide a more accurate estimate.

- *Cooperative fusion*: refers to the fusion of the output of different sensors in which one sensor relies on the output of another sensor/s to make its own observations.

Fusion of data from multiple sensors may be performed at a number of levels from the electronic signal level, to the fusion of sensor data, which has been interpreted to give a higher semantic meaning. Sensors in the sentient object model process raw sensor data and produce a higher-level interpretation and it is this symbolic output that we are interested in fusing. Complementary and competitive fusion are most relevant to sentient objects, where we are trying to both gain a more complete representation of context, and reduce the uncertainty inherent in sensor readings.

Sensor fusion for context-aware sentient objects is different to classical sensor fusion, which is primarily concerned with parameter estimation. With sentient objects, in addition to the use of sensor fusion for the optimisation of measurements, we are also concerned with fusing various seemingly unrelated pieces of information in order to determine context.

2.2.4.3 Bayesian Networks for Sensor Fusion

Sentient objects almost never have access to a completely correct view of their context due to an incomplete or incorrect understanding of their environment for reasons discussed above. In real-world domains, sentient objects can only provide a degree of belief in relevant contexts.

Bayesian theory provides the basis which allows us to reason under uncertainty in the form of probabilities by calculating the probability that a hypothesis (H) is true, given observed evidence (E), or *P(H|E).*

Any complete probabilistic model of a domain must, either explicitly or implicitly, represent the joint distribution - the probability of every possible event as defined by the values of all the variables [5]. Modelling a domain as a set of random variables $X_1,...,X_n$, then $P(X_1,...,X_n)$ denotes their *joint probability distribution* (jpd), or the probability of every possible combination of values for $X_1,...,X_n$. Such a jpd provides complete information about the probabilities of its random variables, but quickly becomes very large. A jpd table for n random variables, each ranging over k distinct values, has k^n entries.

The majority of the time, the variables describing a domain are dependant on each other[1] and the joint probability of two variables may be defined in terms of conditional probabilities

$$P(X_1, X_2) = P(X_1 \mid X_2)\, P(X_2) = P(X_2 \mid X_1)\, P(X_1)$$

This may be rewritten as Bayes' Rule for the conditional probability of two variables as follows

$$P(X_2 \mid X_1) = \frac{P(X_2, X_1)}{P(X_1)} = \frac{P(X_2) P(X_1 \mid X_2)}{P(X_1)}$$

Bayes rule allows us to reason in terms of conditional probabilities. A Bayesian network is a directed acyclic graph in which the nodes represent random variables, and the absence of arcs

[1] Two random variables X_1 and X_2 are independent iff $P(X_1 \mid X_2) = P(X_1)$ or $P(X_1, X_2) = P(X_1)P(X_2)$. Complete independence is a very strong and seldom met requirement

represents conditional independence in that a node is independent of its non-descendants given its parents [15]. Bayesian Networks achieve compactness when compared to jpd's, by factoring the joint distribution into local, conditional distributions for each variable given its parents [5]. For a variable X_i if we denote the parents of X_i as $pa(X_i)$ then $P(X_i \mid pa(X_i))$ denotes the local conditional distribution for X_i given its parents. A Bayesian network allows us to factor the joint distribution over all variables into the product of local terms, giving the full joint distribution over all variables as

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid pa(X_i))$$

An example Bayesian network is illustrated in Figure 3 for five variables. For this example network, $P(A, B, C, D, E) = P(A)\,P(B)\,P(C|A)\,P(D|A, B)\,P(E|D)$. From this example, we can see that provided the number of parents of each node is bounded, the number of parameters required grows linearly with the size of the network, whereas the joint distribution grows exponentially [5].

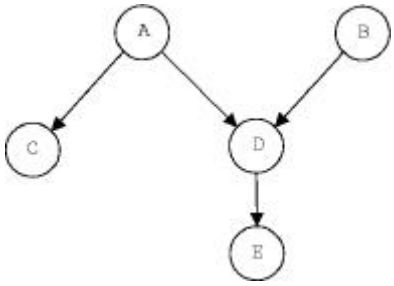


Figure 3: A Bayesian network representing causal inferences among five variables

The arcs in a Bayesian network then represent the conditional probability of the existence of the node being pointed to, given the existence of the node from which the arc originates.

Bayesian networks capture relationships between variables in a system and the dependencies, which exist between them. The networks operate by propagating evidence about variables through the network, according to the relationships encoded therein. In this way, knowledge of the value of a variable allows us to update our belief in the values of all variables in the network.

A Bayesian network for sensor fusion may be constructed by representing the outputs of sensors as inputs into a network, which also contains nodes representing variables, whose value is influenced by the sensor outputs. The uncertainty associated with a sensor reading is then modelled using conditional probabilities associated with causal relationships within the network.

With respect to the sentient object model, we can create a Bayesian network where the **leaf** nodes represent sensors contributing to the determination of the current context of an object. **Terminal** nodes (those nodes without children) represent contexts, whilst intermediate nodes represent the uncertainty values associated with sensors.

Variable	Parent Variable Values		True	False
Person In Room (D)	Temp = 10 – 20	PIR = On	0.6	0.4
		PIR = Off	0.1	0.9
	Temp = 20 – 30	PIR = On	0.9	0.1
		PIR = Off	0.05	0.95
	Temp = 30+	PIR = On	0.4	0.6
		PIR = Off	0.2	0.8
Door Closed (E)	Reed Switch = Closed		0.95	0.05
	Reed Switch = Open		0.05	0.95
Working Hard (F)	Person In Room	Door Closed	0.95	0.05
		¬Door Closed	0.4	0.6
	¬Person In Room	Door Closed	0.3	0.7
		¬Door Closed	0.05	0.95

Table 1: Conditional probability tables for nodes in Figure 3

By way of an example, we consider the case of a sentient office that is equipped with a number of sensors and actuators. The office may exist in any one of a set of contexts, depending on what purpose the office is being used for at a point in time and sensor outputs need to be fused together in order to determine this context. For the purposes of this example, the office is equipped with three sensors and one actuator. The sensors consist of a Passive Infrared (PIR) sensor mounted next to the desk, a reed switch mounted on the door and a temperature sensor. The actuator controls whether the telephone answering machine is set to automatically answer incoming calls. Given the output of the three sensors, we would like to determine, with a bounded degree of certainty, what context the office is in. For instance if there is a person in the office, and the door is closed the office is in the context of being used for hard work and as a result the answering machine should be set to answer all incoming calls so as not to disturb the worker.

We may easily construct a Bayesian network that fuses the output of the three sensors in the sentient office as illustrated in Figure 4. The leaf nodes, shaded grey in the network represent the sensors themselves, whilst the intermediate nodes represent the uncertainty associated with the sensor readings as determined from experimental observation / operational specifications. The probability that the sensor reading is correct is captured in conditional probability tables (cpt) at each node. The cpt's for non-leaf nodes in the network are illustrated in Table 1[2]. Examining the entry for the node **Person in Room**, the table captures the probability that there is actually somebody in the office given the output of the PIR and temperature sensors and is constructed through experimental observation. The reason that we fuse the output of the PIR and temperature sensors when determining if there is somebody in the office is that PIR sensors work by sensing temperature variations and consequently their accuracy is influenced by environmental temperature. Reed switches are not dependant on temperature fluctuations and so there is no causal relationship between the temperature sensor and the reed switch sensor.

[2] These probabilities are for illustrative purposes only and are not derived from empirical measurements.

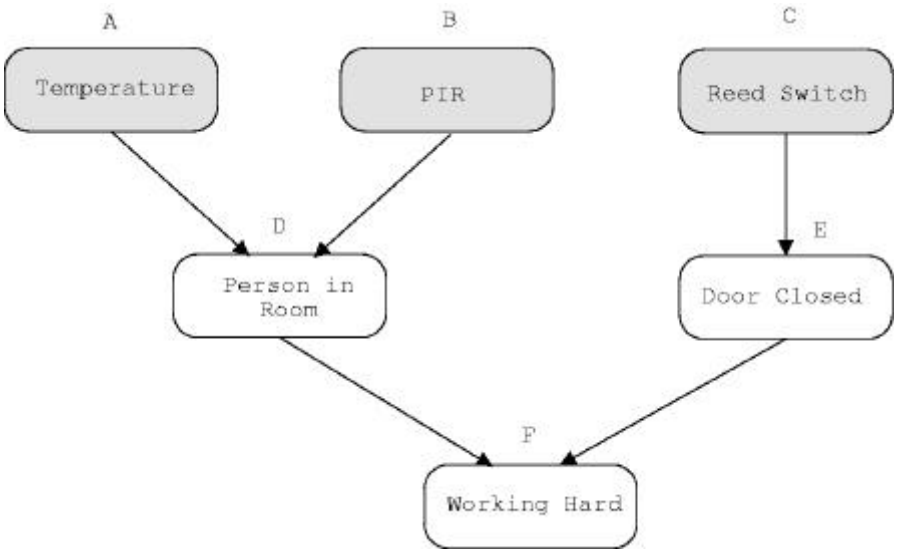


Figure 4: A Bayesian network for fusing the output of three sensors

2.2.4.4 Incorporating Probabilistic Sensor Fusion into the CxBR hierarchy

CxBR is based upon the hypothesis that only a small portion of all available environmental inputs are used to recognise and treat the key features of a situation and that within any given situation, there are only a limited number of things that can realistically take place. The fact that only a small portion of sensory input is relevant at any point in time is used to enhance the effectiveness of the probabilistic sensor fusion scheme by limiting the number of nodes in the Bayesian network in each context.

Sensors in the sentient object model are highly distributed, with changing configurations due to the mobility of sentient objects. In addition, which of a set of sensors are consulted at a particular point in time is highly dependant on the active context at that time. We perform sensor fusion at the level of a context within the CxBR hierarchy. A context defines which sensors are relevant to that context as well as those that must be monitored in order to detect when transition to another context is indicated. A Bayesian network is constructed within each context in order to fuse the fragments of context information obtained from the sensors. The integration of Bayesian network fragments into the CxBR hierarchy is illustrated in Figure 5. This figure shows how each context in the hierarchy is only interested in a subset of the sensor input, and Bayesian network fragments within each context act to fuse the context fragments obtained from the relevant sensors.

In this way the sensor fusion task is modularised and vastly reduced in complexity. For each possible context in the CxBR hierarchy for a sentient object, a sensor fusion module is created which supports the fusion of pieces of context information obtained from relevant sensors. The fusion service is well separated from context acquisition as well as being separated from the inference engine. Such separation permits for the fusion service implementation to be easily replaced without affecting other components.

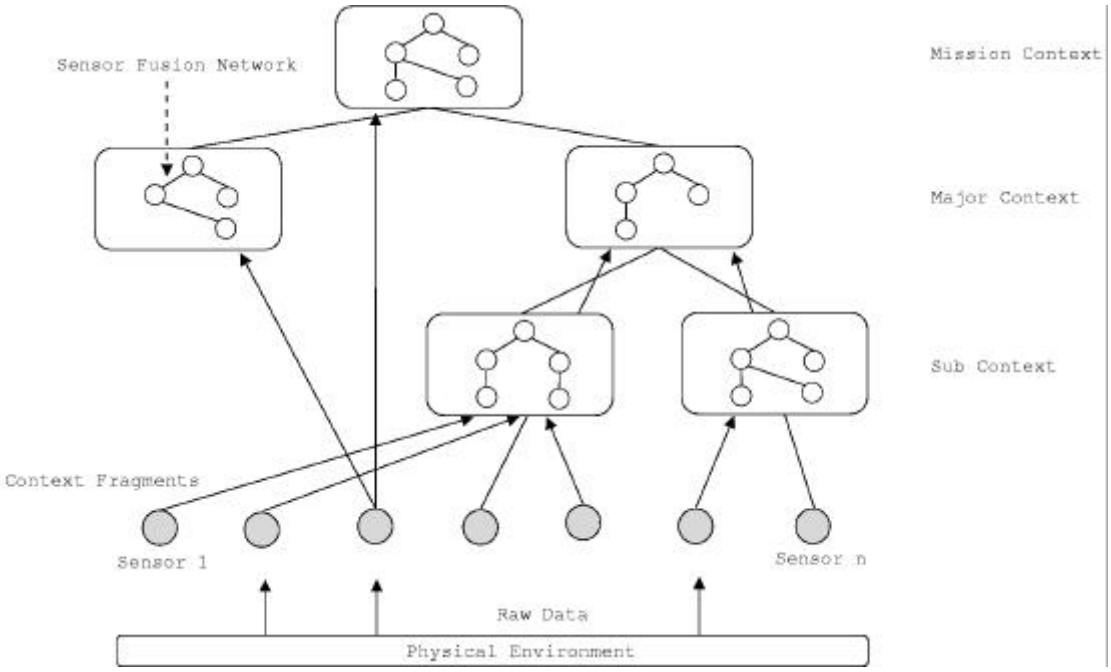


Figure 5: Sensor fusion in the CxBR hierarchy

2.2.5 Conclusions and Future Work

We are developing middleware, based on the sentient object model, to support context-aware applications. This middleware will permit the development of context-aware applications based on environmental perception whilst insulating the programmer from the complexities of physical sensors and their associated uncertainties. Programming such applications will consist of intuitively specifying the CxBR hierarchy and associated Bayesian network structure for fusion of context fragments. We are currently developing a large-scale sentient traffic simulation based on the sentient object model. In addition we are extending existing smart office infrastructure to make use of the sentient object model and its associated CxBR hierarchy based fusion services.

Probabilistic models and particularly Bayesian networks provide us with a means to measure the effectiveness of our derivations of context information within a sentient object given inherently uncertain and noisy sensor data. The quantification of the reliability of a single sensor is a difficult task, but by doing so we are able to provide bounds on the certainty of information derived from the sensor.

2.2.6 Acknowledgements

The work described in this paper was partly supported by the Future and Emerging Technologies programme of the Commission of the European Union under research contract IST-2000-26031 (CORTEX - CO-operating Real-time senTient objects: architecture and EXperimental evaluation). The authors are grateful to Raymond Cunningham for his valuable input.

References

[1] Adrian Fitzpatrick, Gregory Biegel, Siobhan Clarke, Vinny Cahill Towards a Sentient Object Model Workshop on Engineering Context-Aware Object Oriented Systems and Environments (ECOOSE), Seattle WA, USA November 2002.

[2] M.T. Smith SmartCards: Integrating for Portable Complexity IEEE Computer, Volume 31, No. 8, pp. 110-112, August 1998.

[3] B. Clarkson, K. Mase and A. Pentland Recognizing User Context via Wearable Sensors in Proceedings of the IEEE International Symposium on Wearable Computing (ISWC'00), Atlanta, GA, USA, pp. 69-76, October 2000.

[4] Avelino J. Gonzalez, Robert Ahlers Context-Based Representation of Intelligent Behaviour in Training Simulations Transactions of the Society for Computer Simulation International, Vol. 15, No. 4, March 1999.

[5] Judea Pearl Bayesian Networks Handbook of Brain Theory and Neural Networks, MIT Press, 2001.

[6] Anind K. Dey and Gregory D. Abowd The Context Toolkit: Aiding the Development of Context-Aware Applications in Proceedings of Human Factors in Computing Systems: CHI 99. Pittsburgh, PA: ACM Press. pp. 434-441, May 15-20 1999.

[7] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced Interaction in Context in Handheld and Ubiquitous Computing Springer-Verlag, pp. 89-101, 1999.

[8] Paul Castro and Richard Muntz, Managing Context Data for Smart Spaces IEEE Personal Communications, Volume 7: Issue 5, October 2000.

[9] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, Koos Rommelse The Lumiere Project: Bayesian User Modelling for Inferring the Goals and Needs of Software Users in Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, pp. 256-265, July 1998.

[10] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz A Bayesian Approach to Filtering Junk E-Mail AAAI Workshop on Learning for Text Categorization, Madison, Wisconsin. AAAI Technical Report WS-98-05, July1998.

[11] 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) Vienna, Austria, pp. 639-644, 2002.

[12] Huadong Wu, Mel Siegel and Sevim Ablay Sensor Fusion for Context Understanding IEEE Instrumentation and Measurement Technology Conference, Anchorage, AK, USA, 21-23 May, 2002.

[13] A. Schmidt, M. Beigl and H-W Gellerson There is More to Context than Location Computers and Graphics, Volume 23, No. 6, pp.893-901, 1999.

[14] Arnoud Visser Organisation and Design of Autonomous Systems Faculty of Mathematics, Computer Science, Physics and Astronomy University of Amsterdam, August 1999.

[15] James M. Rehg, Kevin P. Murphy, Paul W. Fieguth Vision-Based Speaker Detection Using Bayesian Networks in Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, Ft. Collins, CO, pp. 110--116, 1999.

2.3 Machine Learning

The concern of the field of machine learning research is how to construct computer programs that can learn with past experiences in order to automatically improve performance [Mitchell, 1997]. Machine learning is typically considered as a sub-topic of artificial intelligence and a multi-disciplinary subject inspired by cognitive sciences, computer sciences, pattern recognition, and statistics [Aha, 1995]. In the context of the sentient room demo, machine learning can be actively investigated as a practical method to learn a user's interest, preference, knowledge, goal, habits, etc. in order to adapt the services to the user's individual characteristics [Pohl W. 1996]. The way how multiple people resolve conflicts in the sentient room can also become the target function of the machine learning algorithms, so that the sentient room can make autonomous decision on future conflicts. It is also plausible to apply machine learning algorithms for inducing low-level contexts from sensory data, e.g., inducing how many people in the room from environmental parameters.

The most popular and widely-used machine learning methods are the family of the inductive learning methods, which is based on the following assumption: "Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples" [Mitchell, 1997]. The member of the inductive machine learning family includes decision tree learning, neural network learning, and Bayesian learning, etc.

2.3.1 Decision Tree Learning

Decision tree learning is a method for approximating discrete-valued target functions, and it is robust to noisy data and capable of learning disjunctive expressions. The learned function is represented in the form of a decision tree, which can also be described in some human friendly if-then rules.

A decision tree is created from a training set of tuples, each of which is composed of an instance (which can have several attributes) and a value (e.g., positive or negative) for the target attribute. Most algorithms for the construction of decision trees are based on the concept of a top-down and greedy search through the space of possible decision trees. The famous decision tree algorithms based on the above approach, such as ID3 [Quinlan, J. R. 1986], ASSISTANT [Cestnik et al. 1987], C4.5 [Quinlan, J. R. 1993)] have already been widely used both academically and commercially.

After being constructed, a decision tree can classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. A path from the root to a leaf node corresponds to a rule learned from the training samples. An instance is classified by starting the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. The process is then repeated for the sub-tree rooted at the new node. In general, a decision tree can be interpreted as a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the root to a leaf can be considered as a conjunction of attribute tests, and the tree itself as a disjunction of these conjunctions [Mitchell, 1997].

2.3.2 Neural Network Learning

Neural network learning methods provide a general approach to approximating real-valued, discrete-valued, and vector-valued target functions [Mitchell, 1997]. It is robust to errors in training data and has been successfully applied to many practical problems such as interpreting visual scenes, speech recognition, and leaning robot control strategy.

An artificial neural network is an information-processing paradigm inspired by the way the densely interconnected, parallel structure of the mammalian brain processes information. An artificial neural network is built out of a densely interconnected set of simple units (or nodes), where each unit takes a number of real-valued inputs (possibly outputs of other units) and produces a single real-valued output (which may become the input to many other units). The processing ability of the network is stored in the inter-unit connection strengths (or weights), obtained by a process of learning from a set of training patterns.

The Backpropagation algorithm is the most influential and commonly used neural network leaning technique. In this algorithm, the network consists of a fixed set of units, which are normally organized into multiple layers: input layer, at least one intermediate hidden layer, and output layer. Each unit in one layer is connected with all the units in the above layer, and the algorithm learns the weights of each connection based on the training samples. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs. The weights learned by neural networks are often difficult for humans to interpret.

2.3.3 Bayesian Learning

Bayesian learning is a straightforward learning algorithm that calculates the probability for each candidate hypothesis and selects the most probable posterior hypothesis. It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data [Mitchell, 1997].

The basis of the Bayesian learning is Bayes theorem, which provides a direct method for calculating the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself [Mitchell, 1997]. A highly practical Bayesian learning method is the naive Bayes classifier, where the training data is a set of tuples composed of a conjunction of attribute values and the target function. The target function can take on any value from a finite set. The leaner is asked to predict the target value or classification for anew instance described by the tuple of attribute values. The naive Bayes learning method selects the most probable hypothesis by estimating the various prior probabilities and conditional probabilities, which can be calculated over the training data. Instead of explicit searching through the space of possible hypothesis, the Bayesian learning simply counts the frequency of various data combinations within the training examples.

In the Naive Bayes classifier, the assumption that the attribute values are conditionally independent given the target value might be overly restrictive. In some cases, the attributes in a training data set can be dependent on each other. Bayesian Belief Networks describe the dependence and independence between attributes using a graph. Using this networks can be an intermediate approach between the overall assumption of conditional independence in the naive Bayes classifier and avoiding conditional independence assumptions altogether. The network structure can be defined from prior knowledge or can be learned from the training data set itself [Heckerman, 1996].

2.3.4 Design Consideration

When introducing machine learning methods into the sentient object paradigm, the following issues need to be considered:

• Training Experience: The type of training experience is the first design choice we face. One key attribute is to choose direct or indirect training examples, which provide direct or indirect feedback regarding the choices made by the performance system [Mitchell, 1997]. The degree of controlling over the sequence of the training example by the learning system is the second important attributes. The third attribute is how well the training experience represents the distribution of examples.

• Target Function: It is to determine what will be learned and how this will be used. In the context of the sentient room demo, are we going to learn the high level personal preferences or to infer how many people in the room from the low-level environmental parameters? Or both?

• Representation: We must choose proper representations for the target function as well as the training examples. Standardized representation improves portability and extensibility of the system.

• Learning Algorithm: Now we can determine to use which machine learning algorithm for the specific problem according to our selection criteria: some algorithms might be more efficient for certain application domains; some might be easier for human to understand and interpret; some might be more robust than other on dealing with the uncertainty of the input data, etc. All these attributes could become the reason of choosing certain learning algorithm.

References

Aha D. W. (1995) Machine Learning, A tutorial presented at the 5th International Workshop on Artificial Intelligence & Statistics.

B. Castaic, I. Koromiko, and I. Brake (1997), Assistant 86: a knowledge-elicitation tool for sophisticated users. In I. Brake and N. Larva (eds.), Progress in Machine Learning -- Proceedings of the Second European Working Session on Learning (EWSL87), Wilson, UK: Sigma Press, 31-45.

Heckerman D. (1996) A Tutorial on Learning With Bayesian Networks, Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division.

Mitchell T. M. (1997) Machine Learning, McGraw-Hill, ISBN 0-07-115467-1.

Pohl W. (1996) Learning about the User – User Modeling and Machine Learning, Workshop in ICML'96 on Machine Learning meets Human-Computer Interaction.

Quinlan, J. R. (1986), Induction of decision trees, Machine Learning, 1(1), 81-106.

Quinlan, J. R. (1993), C4.5: Programs for Machine Learning, 2(3), 229-246.

Chapter 3: Event-Based Communication Model

3.1 An Event Model for Real-Time Systems in Mobile Environments

René Meier[1], Jörg Kaiser[2], Barbara Hughes[1], Cristiano Brudna[2], Vinny Cahill[1]

[1]Department of Computer Science, Trinity College Dublin, Ireland
[2]Department of Computer Structures, University of Ulm, Germany
{Rene. Meier, Barbara. Hughes, vinny.cahill}@cs.tcd.ie,

{kaiser, Christiano.brudna}@informatik.uni-ulm.de

ABSTRACT

This paper describes an event model that has been designed to address the predictability requirements of applications operating in mobile environments based on hierarchically structured WAN-of-CANs network. The event model supports an event channel concept for modeling the guarantees provided by the underlying, heterogeneous communication infrastructure. The networks that comprise such a WAN-of-CANs may provide fundamentally different degrees of quality of service and as a result can be viewed as zones within which certain guarantees can be enforced. Event channels operating in CAN-based subnetworks with typically strong timing behavior may support hard temporal and reliability attributes whereas channels interconnecting these subnetworks using wireless networks support weaker timing attributes.

Keywords

Event-based middleware, timely event delivery, mobile computing, CAN networks, wireless networks.

3.1.1 INTRODUCTION

Advances in information technology encourage new classes of applications that are based on a large number of networked components acting autonomously in response to a myriad of sensors and actuators to assess and control aspects of the environment. Examples range in telematics, traffic management or home automation to name a few. To a large extent, such systems operate proactively and independently of direct human control driven by the perception of the environment and the ability to organize respective computations dynamically.

Conference '00, Month 1-2, 2000, City, State.

The challenging characteristics of these applications include sentience and autonomy of components, issues of responsiveness and safety criticality, geographical dispersion, mobility and evolution.

Such systems require a degree of co-operation, adaptability, extensibility and reliability that is not available today. The problem is to provide a communication and interaction scheme that supports a large-scale many-to-many communication relation typical for these applications. Additionally, it should be possible to seamlessly disseminate relevant information generated by deeply embedded controllers to all interested entities in the global network.

The contribution of the paper is a model for a real-time event system on top of a heterogeneous communication system. Such a system may be composed from networks with largely different quality characteristics ranging from highly predictable real-time buses to ad-hoc wireless networks in which mobility of nodes result in frequent link failures. The focus of the paper is on the quality aspects of communication, explicitly providing a system architecture that reflects a realistic network model in large-scale cooperative applications. It presents abstractions to encapsulate network properties and make them assessable on the level of the event system. Related work on event systems does not allow to explicitly model quality properties resulting from heterogeneity [1-3], or it deals with mechanisms that mask network heterogeneity, thereby neglecting respective real-time and quality issues.

3.1.1.1 Overview

The event-based communication model represents a paradigm for middleware that asynchronously interconnects the components that comprise distributed applications [4]. Event-based middleware has been recognized as being well suited to addressing the requirements of mobile application [5-8] and to connecting the components in control applications with timeliness requirements [1-3]. However, to date event models have not been designed to offer sufficient levels of dependability for real-time systems while supporting mobile application components.

This paper presents an event model addressing the predictability requirements of large-scale applications operating in mobile environments based on heterogeneous communication infrastructures. Our event model defines an abstract network model and provides mappings for the envisaged network types. These networks may include Controller Area Networks (CAN), Local Area Networks (LAN), and Wide Area Networks (WAN), especially those based on wireless technology. The underlying communication infrastructure typically consists of a hierarchically structured WAN-of-CANs network, an internetwork whose subnetworks will typically be CANs that are interconnected by means of wireless LANs and WANs. The networks that comprise such a WAN-of-CANs typically provide fundamentally different degrees of Quality of Service (QoS), ranging from CAN networks with strong timing and reliability behavior to wireless network supporting mobile application components and as a result providing weaker guarantees. We regard individual networks as QoS containers, called zones, within which given degrees of predictability in terms of timing and reliability can be enforced.

Our event model provides a programming model based on the concept of event channels. A number of event channel classes with different temporal and reliability attributes are supported for integrating the real-time aspects into the event channel model. Depending on the guarantees available from the underlying network, these event channels can be mapped to certain QoS zones. Generally, channels can be associated with zones providing equal or stronger guarantees.

In order to address the requirements of real-time systems, we support classes of hard, soft, and non real-time event channels. Hard real-time event channels are considered to meet all temporal requirements under the specified fault assumptions. Soft real-time event channels are scheduled according to deadlines. Nevertheless, these deadlines are not guaranteed during transient overload conditions. Non real-time event channels are typically used for best-effort event delivery without any specified timeliness requirements. In general, any class of event channel may be mapped to zones representing CAN-based subnetworks whereas zones including wireless network can only support guarantees sufficient for soft and non real-time event channels. However, we envisaged using a predictable medium access protocol, such as the Time-Bounded Medium Access Control (TBMAC) [9] protocol, in order to provide soft real-time deadlines with a high probability for event channels propagating events in wireless networks.

3.1.1.2 Paper Organization

The remainder of this paper is structured as follows: Section 3.1.2 presents the system architecture including the underlying network infrastructure and the classes of event channels. In section 3.1.3, we discuss arising issues when applying real-time to a WAN-of-CANs network, especially those based on wireless technology. Section 3.1.4 focuses on the programming model supported by our event model and section 0 outlines the communications architecture. Section 0 concludes this paper by summarising our work and outlining the issues that remain open for future work.

3.1.2 SYSTEM ARCHITECTURE

When striving for a real-time event system, the communications architecture substantially affects the temporal and reliability properties of event dissemination. Thus, in quality terms the underlying network is not transparent for the application using the event system. The goal for our event system is to express the quality requirements of event dissemination on the abstraction level of events rather than to resort to lower network details. The middleware then automatically maps these requirements to the underlying network. Hence, we will briefly introduce the general assumptions about the network architecture.

3.1.2.1 General Network Structure

From an application point of view, the event system should support cooperating autonomous entities, such as cars, robots, and people wearing computers and moving through a smart environment. As an example consider autonomous mobile robots which may be equipped with all kinds of smart sensors, such as lasers, radars, cameras, navigational sensors, and chemical sensors, to achieve an adequate perception of their environment. The necessary highly reactive behaviour of such an individual robot has to be achieved by a tight cooperation of the sensor / actuator system over some special purpose network inside the robot, called a CAN (Controller Area Network in a broad sense). Additionally, the vehicles may communicate via a wireless link carrying out joint tasks. At an abstract layer, this is modelled as islands of tight control cooperating via a WAN-of-CANs structure [10]. We assume a certain guaranteed level of predictability as an intrinsic property of a CAN. A CAN therefore constitutes a zone of coherent QoS provision.

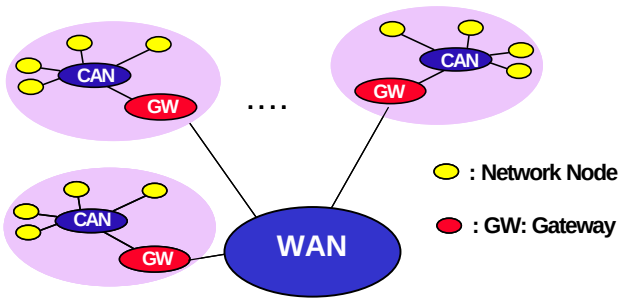


Figure 1. The WAN-of-CANs network architecture.

The general structure depicted in Figure 1 is recursive in a sense that a CAN may be composed from a hierarchy of networks itself. This issue is not further discussed in this paper. It should only be noted that each of the interconnected subnets constitutes a zone in which a certain QoS is defined. The gateways have multiple functions. Firstly, they route events over network boundaries by maintaining an event interface that is

transparent with respect to functional properties throughout the network. For non-functional quality attributes they provide the needed awareness about the network properties. Thus, they have to marshal events depending on the underlying abstract network infrastructure, which is described below. Secondly, they constitute filters that allow to scope events according to their context and their quality attributes as introduced in section 3.1.4. A realisation of this structure for cooperating mobile robots internally equipped with a CAN-Bus, which is extended via gateways to a wireless LAN for event dissemination between the robots is presented in [11]. Furthermore, a middleware service that has been designed for interconnecting mobile application components using ad hoc wireless local area networks has been described in [12].

3.1.2.2 Abstract Communication Layers

There is a trade-off between the predictability of communication and the needed resources. At the safe end, all communication is statically planned and resources have to be assigned anticipating worst-case load and failure assumptions. However, this may only be required for a small number of highly critical services. In fact, critical system services as could use such highly predictable links to meet its temporal and reliability requirements, e.g. tight sensor / actuator control loops e.g. for crash avoidance or motor and brake control. In most cases less critical events have to be accommodated by the communication system, which also allow more dynamic system behaviour. However, also for these events, temporal parameters may be needed, e.g. the specification of when an event should be delivered or how long the event is valid. The different requirements are reflected by event channel classes with different properties. On the architectural level, we distinguish three layers roughly depicted in Figure 2.

On the event layer, the main abstractions are events and different classes of event channels. As described in section 3.1.4, this layer enables the application to specify channels with different QoS properties. Mapping the abstractions of the event layer directly to the underlying network is a tough challenge because the usual abstractions on the network layer are low-level messages. Hence, this layer does not match the requirements of group communication, subject-based addressing or the QoS specifications defined for channels.

Therefore, an abstract network layer is introduced enriching the properties of the raw network (this may not just be a physical network, but a specific MAC-layer or even a higher OSI level) with additional properties and communication services such as reliable broadcast, group communication, and temporal guarantees for message dissemination. This separation of concerns supports modularity and allows for an easier adaptation of the event layer to networks with widely differing characteristics.

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 support highly safety critical hard real-time event channels if the abstract network layer cannot provide the

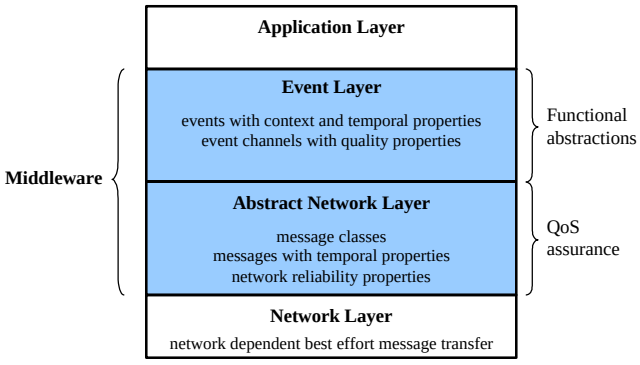


Figure 2. Architectural layer.

required guarantees. Therefore, the event channels supported by the event system are dependent on the zones in which the respective protocols can be provided. As an example, [13] describes the protocols and services of the abstract network layer for a particular CAN-Bus.

3.1.3 REAL-TIME AND MOBILITY

Previous research in real-time event-based communication [1, 3] has focused on wired networks assuming a static number of location fixed nodes and a known fault hypothesis. Transmission schedules for avoiding collisions are typically planned statically and the correctness of these schedules regarding timing conflicts and temporal overlaps are verified off-line. However, such an approach would be inappropriate for achieving real-time guarantees for wireless environments, especially when using ad hoc networks. The characteristics of wireless networks that render such an approach unsuitable include highly dynamic connections, location-dependent coverage and contention, and its susceptibility to the physical environment.

3.1.3.1 Dynamic Connections

Wireless network connections between mobile nodes are typically established dynamically due to the unpredictability of node movements. Maintaining accurate network information in such an environment, which is essential for enforcing delivery guarantees, requires the propagation of updates reflecting the actual network topology. The frequency for disseminating these updates relates directly to the level of node mobility and consequently results in additional, potentially unpredictable communication and computational load.

Link failures are the norm rather that the exception in wireless networks. Links may fail as a result of various reasons including individual node failures, for example due to low battery power, physical obstacles blocking a transmission, and nodes moving out of transmission range. Significantly, link failures may result in network partitions where groups of node operate independently of each other. Adapting to link failures relies on timely and accurate topology updates, which may be affected by the unpredictability of the network and by the presence of

partitions. Updates may be confined to certain partitions, which implies partial knowledge of the network topology.

Timeliness guarantees for event delivery rely on a transmission schedule that reflects the priorities and deadlines associated with events and on the precision of such a schedule, which derives from the accuracy of the available typology information. Hence, providing guarantees with a high probability is hard in wireless environments as the required topology information is likely to be out of date or even incomplete. Moreover, it may be impossible to deliver an event to all subscribers in the presence of partitions.

3.1.3.2 Location-Dependent Coverage And Contention

Coverage can be defined as the geographical area to which a particular node can send messages using single-hop or multi-hop communication. The area covered by the wireless transmitters of mobile nodes typically changes over time depending on the movements of individual nodes. Consequently, coverage describes the set of nodes, which is likely to change over time, to which a specific node can propagate events at a given time. Kanodia et al. [14] propose a protocol for ordered event scheduling in ad hoc wireless networks. This work focuses on topologies in which all nodes reside within single-hop transmission range of each other and as a result assumes that these nodes have the same information describing the network topology. Tobagi and Kleinrock [15] as well as Kanodia et al. [14] discuss topologies where nodes use multi-hop communication when propagating messages. Such nodes may have different probabilities of accessing a wireless communication channel and different throughput capabilities, depending on properties such as spatial reuse and contention degree. Unequal channel access is detrimental to achieving distributed agreement and results in unpredictable transmission delays, which leads to unbound delivery latency.

Wireless transmissions in ad hoc networks are broadcast through a physical communication channel shared by a group of nodes. This implies that the contention level of a specific channel depends on its location. In order to avoid collisions at a certain location, nodes sense an ongoing transmission and restrain from sending until the channel becomes available. Hence, nodes must exploit a means to arbitrate spatial contentions between the transmissions in their spatial locality. Luo and Lu [16] propose a fair channel access model that ensures coordination among spatially contending transmissions. However, this work assumes that individual transmissions are decoupled from each other and that scheduling can be performed independently for each transmission. This cannot be assumed in applications comprising highly mobile nodes. Moreover, neither Tobagi and Kleinrock [15] nor Luo and Lu [16] address temporal or reliability constraints, which are essential for providing end-to-end delivery guarantees.

3.1.3.3 Susceptibility To The Physical Environment

The physical environment has a critical impact on wireless communication. Many physical objects such as vehicles and building as well as people themselves may represent obstacles that effect transmissions. Obstacles impede wireless signals and result in a reduction of the available signal strength. Consequently, they may lead to packet loss as well as to unpredictable transmission delays and jitter. The dynamic aspect of parts of the physical environment causes unpredictable occurrences and durations of such impeding effects. This results in significant fluctuations between intervals of high and low connectivity. Significantly, this unpredictable, dynamic behavior of the physical environment causes variations in the quality of service available from the network.

3.1.4 THE PROGRAMMING MODEL

As a consequence of the interaction with the physical world, the pace of interactions and of the computational progress in the distributed system is dictated by the progression of real time and the properties of the environment. To support coordination of actions, cooperation between mobile entities and fusion of elementary events in the sentient object model [17], the events must be disseminated in a predictable way.

An event encapsulates the description of an observation about the change in the environment or the system. Hence, it is related to an occurrence in the real world, e.g. observed by a sensor, or an in the control system itself, e.g. generated by some control program. An event is an instance of an event type, which is characterized by subject, attributes, and content:

event := <subject, attribute_list, content >.

The subject is attached to an event as a tag to explicitly identify the event's type and enable basic filtering and some form of type checking on event dissemination. An event is further characterized by a set of functional and non-functional attributes. Functional attributes are related to the application relevant context in which an event has been generated and may include information like location, range of dissemination (relative location or proximity) and time of occurrence, or system related information like a mode of operation or a network zone in a WAN-of-CANs architecture. Non-functional attributes represent intrinsic properties and quality aspects of an event. These may be deadlines necessary to express temporal requirements of event dissemination or a validity interval according to an aging function. A deadline is used to schedule an event in the communication system and define priority relations to other events. The validity interval defines an application related interval that may go beyond the deadline when the usefulness of the event expires. It allows a certain tolerance of transient overload when the event cannot be delivered within the deadline but eventually to discard an event and purge the system from outdated events. This is

particularly important in a real time event system to prevent the already outdated events to still compete for resources with the actual events. Finally, the content of an event carries the data that is represented as a structured set of functional parameters. The content is accessible by specific methods.

Predictability incorporates the timely delivery of events under anticipated load and failure conditions. The nature of events may range from a safety-critical event signaling that a crash of a mobile vehicle with an obstacle is about to happen to the dissemination of non-critical events describing a room temperature or illumination intensity that have less demanding predictability requirements. As a matter of fact, there is a trade-off between the resources expended and the degree of predictability. Consequently, the respective requirements for the underlying abstract network may range from synchronous reliable broadcasts [11, 18] to best effort communication.

To facilitate the specification of the particular requirements for event dissemination the notion of event channels is introduced. An event channel encapsulates properties of the underlying communication system and allows specifying quality attributes on an abstraction level where it is assessable to an application programmer. The benefits are twofold: Firstly, a check can be performed whether the non-functional attributes of an event match to the quality attributes defined for the event channel. This allows early timing failure detection. Secondly, predictability requires that resources are available when they are needed. An event channel can be established and the necessary resources can be assigned by the middleware before an event has to be disseminated. The general form of an event channel representation is given by:

event_channel := <subject, attribute_list, handlers >

In contrast to the attributes of an event, which describes the properties of a single individual occurrence of an event, the attributes of the event channel abstract the properties of the underlying communication network and dissemination scheme. Therefore the attributes define quality properties and include e.g. latency, dissemination constraints and reliability parameters. The subject of the event channel must match the subject of the event that is disseminated through the channel. The "handlers"-field allows specifying notification and exception handlers for the event channel.

According to the need in most real-time systems, particularly, if we assume the network architecture introduced in section 3.1.2, event channels with different timeliness and reliability properties must be supported. We distinguish three event channel classes: Hard Real-Time Event Channels (HRTEC), Soft Real-Time Event Channels (SRTEC) and Non Real-Time Event Channels (N7RTEC). A HRTEC offers rigorous guarantees for discrete control based on sporadic events as well as for continuous control requiring periodic events like sensor readings and control feedback. For sporadic events a maximum latency will be guaranteed while for periodic events the goal is to achieve a low period- and latency-jitter. The guarantees are maintained under an anticipated number of omission failures. Events published to a SRTEC are scheduled according to the Earliest Deadline First (EDF) algorithm. As outlined below, deadlines may be missed in situations of transient overload or due to the arbitrary arrival times of messages. Finally, a NRTEC disseminates events that have no timeliness requirements.

The transport of events through a hard real-time event channel (HRTEC) is synchronous and reliable. The properties of a HRTEC are defined by: 1.) a known upper bound for the transport latency, i.e. the interval between the point in time when an event message becomes ready and its delivery; 2.) a known upper bound for the latency jitter, i.e. the variance of the transport latency; 3.) a known upper bound of the period jitter for periodic events, i.e. the variance on the period; 4.) a fault assumption under which the properties 1.)- 3.) are valid. In order to offer such properties, a HRTEC transparently handles redundant transmissions of events and guarantees that the respective publisher has a privileged access to the communication network. A HRTC s is based on a highly predictable abstract network, e.g. enabling the reservation of network resources. Highly predictable and reliable protocols such as [9, 18, 19] can provide the respective quality of service in the abstract network layer.

Soft real-time events have timeliness requirement that are expressed by deadlines and validity intervals. Thus, a SRTC has to reflect these properties. Soft real-time event messages become ready at any time and are scheduled according to their transmission deadlines. Different from HRTCs, SRTECs do not use reservations. The transmission deadline is defined as the latest point in time when a message has to be transmitted and events are scheduled according to an earliest deadline first algorithm. However, because a message can not be interrupted during its transmission and messages may become ready at arbitrary points in time, EDF will not always take the right scheduling decisions (only a clairvoyant scheduler would be able to do so) and situations of temporal conflicts and transient overload may occur. In theses situations, messages will still be transmitted at a later time in a best effort manner. An SRT event message eventually will be discarded if its transmission time is delayed beyond its temporal validity. The expiration time is an application specific parameter, which may be defined according to some value function.

NRTCs are used for events that do not have timeliness requirements. They are primarily intended for configuration and maintenance purposes. While HRTC and SRTC disseminate events of restricted length to meet the responsiveness requirements of real-time systems, NRTC may transfer bulk data.

3.1.5 THE COMMUNICATIONS ARCHITECTURE

The design of the communications architecture is motivated by our approach to defining an abstract network model describing the characteristics of the underlying WAN-of-CANs architecture and by our programming model based on our concept of event

channels with different timeliness and reliability properties. We employ a mechanism for enforcing hard real-time event channel properties in CAN networks and another mechanism based on bounding the propagation range of events for providing soft real-time guarantees with high probability in wireless networks.

3.1.5.1 Mapping Event Channels To Zones

Applications may comprise numerous components representing real-world objects that may be mobile and distributed over a large geographical area. Such components are typically location aware and depending on their location may interact using different parts of the underlying WAN-of-CANs network architecture. Our abstract network model describes the characteristics of such a network infrastructure dividing it into zones reflecting the available quality of the network service. Applications may define various event channels disseminating events with different temporal and reliability properties. Depending on the guarantees available from the underlying network, these event channels can be mapped to certain QoS zones. Generally, event channels can be associated with zones providing equal or stronger guarantees. For example, a traffic management application may include vehicles interacting trough wireless networks in order to exchange information on the current traffic situation thereby contributing to better driver awareness and consequently to safer driving. Such information may include an accident notification disseminated by a broken-down car to approaching vehicles. Various components representing intra vehicle objects, such as breaks, accelerator, and speedometer, and lights, might communicate using a CAN-based network. Such inter-vehicle and intra-vehicle communication may use different event channel classes for interconnecting their respective components. Hard real-time event channels may be mapped to a zone incorporating intra vehicle components whereas event channels with weaker delivery guarantees may be mapped to a zone comprising components using wireless communication.

Multiple event channels may be mapped to a particular zone sharing the available resources. Such event channels typically disseminate events describing different information and may support different properties. For example, intra-vehicle communication may include a hard real-time event channel disseminating events on behalf of breaks and accelerator and a non real-time event channel controlling the lights of the vehicle.

Event channels may connect components across multiple parts of the network architecture and as a result may be mapped to multiple zones. The properties that can be enforced by such an event channel depend on the level of QoS provided by each of the zones involved. An event channel can be associated with multiple zones if every zone involved provides equal or stronger guarantees. Moreover, an event channel may only operate across the boundaries of multiple zones if the underlying networks are connected through designated gateway components. Such gateways act as producer and consumer of events on either side of the networks they connect. Events received on one side are disseminated on the other side and vice versa. In addition, gateways allow applications to specify network specific mapping of event data and attributes and handle implicit attributes, such as location. A gateway may use a location service to retrieve its own location and may append this location information to events generated by nearby nodes lacking direct access to a location service. For example, a gateway located in a vehicle may connect a CAN-based intra-vehicle network and a wireless inter-vehicle network. It may attach its own location to events generated by nodes on the CAN network as these reside close to the gateway and maintain their location relative to the gateway.

3.1.5.2 Mapping Event Channels To CAN Networks

Event channels may support hard real-time guarantees in zones represented by special CAN-networks. We exploited the specific mechanisms of a CAN-Bus, which is popular in the automotive industry, to design an abstract network layer that can support different real-time guarantees for the delivery of messages. The focus of this work has been on using the hardware supported, lower level priority mechanisms of the CAN-Bus to schedule messages according to the suggested real-time classes. The abstract network layer thus implements reliable hard real-time message delivery and also the weaker forms of guarantees. On top of this abstract network layer a publisher/subscriber protocol has been devised [3] for real-time control applications providing all real-time event channel classes introduced earlier. For a detailed description of mapping the real-time channels to this different message classes the reader is referred to [13].

3.1.5.3 Mapping Event Channels To Wireless Networks

As we have discussed in section 3.1.3, enforcing hard real-time guarantees in wireless environments in general and in ad hoc networks in particular is problematical due to the dynamic nature of these networks. However, we argue that soft real-time event channels can be supported in zones that contain wireless networks. Significantly, we envisage using a set of techniques in order to provide soft real-time guarantees with a high probability.

We propose to bound the propagation range of events in wireless environments [12] by defining event channel attributes that describe geographical areas within which events are valid. Such proximity-based event dissemination represents a natural way to limit the scope of an event channel, thereby allowing entities to interact based on their current location. An example scenario illustrating such behavior might include a broken-down car disseminating an accident notification to vehicles in its vicinity.

Moreover, we envisaged using a predictable medium access protocol, such as the light-weight, location-aware, atomic multicast protocol for Time-Bounded Medium Access Control (TBMAC) [9], when propagating event notifications. The TBMAC protocol is based on time-division multiple access with dynamic but predictable slot allocation and has been designed for use in multi-hop ad hoc networks. It provides, with high probability, time-bounded access to the wireless medium that can be exploited by event channels with guaranteed response time requirements.

3.1.6 CONCLUSIONS

We have presented an approach to incorporating the topology of a heterogeneous communication infrastructure into an event-based programming model. We have described how an event model may address the predictability requirements of applications operating in mobile environments based on hierarchically structured WAN-of-CANs network. An abstract network model reflecting the fundamentally different levels of quality of service available from the subnetworks that comprise such a WAN-of-CANs network defines QoS containers that can be viewed as zones within which certain guarantees can be enforced. Our programming model is based on a concept of event channels supporting a number of event channel classes with different temporal and reliability attributes. Depending on the guarantees available from the underlying network, these event channels can be mapped to certain QoS zones. Generally, event channels can be associated with zones providing equal or stronger guarantees.

We have introduced two prototype implementations of our event model. The CAN version enforces hard-real time guarantees in CAN-based subnetworks using lower level mechanisms of the CAN-Bus. The LAN version uses techniques, including geographical bounding of the event propagation range and predictable medium access, to provide soft real-time guarantees with a high probability in wireless networks.

Although we have discussed and addressed some of the fundamental issues arising when applying event-based programming to real-time systems in mobile environments, certain issues remain open for future work. We are currently investigating means for applications to program entities acting as gateway components. Approaches, for example based on exploiting rule-based programming models, have to be provided for specifying network specific mappings of event data and attributes. Furthermore, our QoS framework needs to be extended in order to incorporate an access control mechanism for proactively managing the number of entities using a specific event channel.

3.1.7 ACKNOWLEDGMENTS

The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET programme of the Commission of the EU under research contract IST-2000-26031 (CORTEX).

REFERENCES

[1] 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). Atlanta, Georgia, USA: ACM Press, 1997, pp. 184-200.

[2] R. Rajkumar, M. Gagliardi, and L. Sha, "The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation," in Proceedings of the IEEE Real-time Technology and Applications Symposium, 1995.

[3] 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, 1999.

[4] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications," IEEE Computer, vol. 33, pp. 68-76, 2000.

[5] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS," IEEE Transactions on Software Engineering (TSE), vol. 27, pp. 827-850, 2001.

[6] Y. Huang and H. Garcia-Molina, "Publish/Subscribe in a Mobile Environment," in Proceedings of the Second ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe'01). Santa Barbara, CA, USA, 2001, pp. 27-34.

[7] R. Meier, "Communication Paradigms for Mobile Computing," ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), vol. 6, 2002.

[8] H.-A. Jacobsen, "Middleware Services for Selective and Location-based Information Dissemination in Mobile Wireless Networks," presented at Advanced Topic Workshop on Middleware for Mobile Computing (IFIP/ACM Middleware 2001), Heidelberg, Germany, 2001.

[9] R. Cunningham and V. Cahill, "Time Bounded Medium Access Control for Ad Hoc Networks," in Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02). Toulouse, France: ACM Press, 2002, pp. 1-8.

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

[11] J. Kaiser and C. Brudna, "A Publisher/Subscriber Architecture Supporting Interoperability of the CAN-bus and the Internet," in Proceedings of the IEEE International Workshop on Factory Communication Systems (WFCS 2002). Västeras, Sweden, 2002.

[12] 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). Vienna, Austria, 2002, pp. 639-644.

[13] J. Kaiser, C. Brudna, and C. Mitidieri, "A Real-Time Event Channel Model for the CAN Bus," in Proceedings of the Eleventh International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2003). Nice, France, 2003.

[14] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Ordered Packet Scheduling in Wireless Ad Hoc Networks: Mechanisms and Performance Analysis," in Proceedings of ACM MOBIHOC 2002. Lausanne, Switzerland, 2002.

[15] F. A. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution," IEEE Transactions on Communications, vol. 23, pp. 1417-1433, 1975.

[16] H. Luo and S. Lu, "A Topology-Independent Fair Queuing Model in Ad Hoc Wireless Networks," in Proceedings of the IEEE International Conference on Network Protocols (ICNP 2000). Osaka, Japan, 2000.

[17] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill, "Towards a Sentient Object Model," presented at Workshop on Engineering Context-Aware Object Oriented Systems and Environments (OOPSLA/ECOOSE'02), Seattle, Washington, USA, 2002.

[18] F. Cristian, "Synchronous Atomic Broadcast for Redundant Broadcast Channels," The Journal of Real-Time Systems, vol. 2, pp. 195-212, 1990.

[19] J. Kaiser and M. A. Livani, "Achieving Fault-Tolerant Ordered Broadcasts in CAN," in Proceedings of the Third European Dependable Computing Conference (EDCC-3). Prague, Czech Republic, 1999.

3.2 Towards real-time event-based communication in mobile ad hoc wireless networks

Barbara Hughes and Vinny Cahill
Department of Computer Science, Trinity College, Dublin, Ireland
{barbara.hughes, vinny.cahill}@cs.tcd.ie

ABSTRACT

Most previous work on real-time event-based communication has assumed static, infrastructure-based networks. The underlying assumption of this work is that application components are stationary and that a fixed network infrastructure exists to facilitate communication [1]. Mobile ad hoc wireless networks comprise a number of mobile nodes connected by wireless links forming arbitrary time-varying wireless network topologies without using any infrastructure or administrative support. This highly mobile, dynamic network is not suitable to the static design assumptions of infrastructure networks.

In this paper we propose a conceptual model based on predictive techniques. Our model is designed to alleviate the impediments for real-time event-based communication that are characteristic in a mobile ad hoc wireless environment.

The model we propose is the first to directly address the issue of achieving timeliness and reliability considerations for real-time event-based communication in dynamic mobile ad hoc wireless networks. In this paper we describe the impediments imposed by ad hoc wireless networks on real-time event-based communication, and propose a high-level model to reduce their impact.

3.2.1 INTRODUCTION

The heterogeneity and inherent loose coupling that characterizes applications in a wireless ad hoc network promote event-based communication as a natural design abstraction for a growing class of software systems [1]. The event-based communication model is well suited to addressing the requirements in the wireless mobile computing domain [2]. In this domain the infrastructure or the ad hoc network model may be utilized for wireless communication. Infrastructure wireless networks use access points to mediate communication between mobile application components. Mobile ad hoc wireless networks comprise a number of mobile nodes connected by wireless links forming arbitrary time-varying wireless network topologies without using any infrastructure or administrative support. Ad hoc wireless networks are self-creating, self-organizing and self-administering [3].

The event-based communication model has lead to the development of several middleware services that use the event paradigm as a high-level communication abstraction. The underlying assumption of these services is that application components are stationary and that a fixed network infrastructure exists to facilitate communication [1]. The complexities introduced by the mobile ad hoc wireless model, for example dynamic and instantaneous topology changes, are not considered. For event-based communication to scale to mobile ad hoc wireless networks it is important that their design is not based on many of the assumptions made for fixed infrastructure networks, such as low latency, abundant bandwidth, homogeneous platforms, continuous and reliable connectivity and most importantly centralized control [1].

With the increased research in ad hoc networks in recent years new application domains such as inter-vehicle communication and communication between mobile robots have evolved. Timely communication is essential to allow applications in these domains to be realized. The real-time event-based communication paradigm has been recognized as an appropriate high-level communication scheme to connect autonomous components in large distributed control systems [4], but can the existing models extend to real-time event communication between mobile nodes in a dynamic wireless ad hoc environment?

In this paper we discuss the characteristics of ad hoc wireless networks. We identify how these characteristics impede real-time event-based communication in the ad hoc wireless domain. We propose a conceptual model based on predictive techniques to alleviate the impediments characteristic in a mobile ad hoc wireless environment.

In the next section we review the assumptions upon which existing real-time event-based communication models rely. We pay particular attention to the extent to which these assumptions are applicable in the ad hoc wireless domain. The following section describes the limitations on real-time event-based communication due to the characteristics of mobile ad hoc wireless networks and is followed by a description of our conceptual model that uses prediction to overcome these ad hoc wireless limitations. We finish the paper with some conclusions and a discussion of future work.

3.2.2 ASSUMPTIONS IN FIXED INFRASTRUCTURE NETWORKS

For real-time event-based communication models to scale to mobile ad hoc wireless networks it is important for their design not to be based on many of the assumptions made for infrastructure-based networks, both wired and wireless. In this section we review the common assumptions of some real-time event-based communication models for infrastructure networks and discuss their suitability in the mobile ad hoc wireless domain.

- Network-wide services and intermediary middleware components are available: The CORBA Event Service [5], and its extension the CORBA Notification Service [6] allow application components to use event-based communication, in addition to the core functionality of the Object Request Broker (ORB). The CORBA Notification Service provides QoS capabilities such as event reliability, event priority and timed event delivery. Both event models use a single mediator called an event channel, through which all event data is disseminated. In the TAO Real-time Event Service [7],[8], the event channel plays the same intermediary role. However the TAO real-time event channel is extended to include prioritized event scheduling, event dispatching and event filtering. In these models accessibility to the event channel is critical for all entities participating in event-based communication. In the Real-time Event Channel Model for the CAN-Bus [9], three event channels are distinguished: hard real-time event channels (HRTEC), soft real-time event-channels (SRTEC) and non real-time event channels (NRTEC). Each event channel supports different timeliness and reliability properties.

  In infrastructure networks intermediate components such as event channels or event dispatchers [1], [2] are often run independently and remotely. This topology has limited practical implementation in ad hoc wireless networks. A serious impediment of ad hoc wireless networks is the limited area that can be covered by mobile application components using a wireless transmitter. Event-based communication using an intermediary middleware component requires all entities in a system to be able to communicate with it at any given time. In an ad hoc wireless network entities may be distributed over a potentially large geographical area and thus are unlikely to be able to maintain a permanent communication link to the intermediate [10]. This impediment of wireless ad hoc networks necessitates the omission of intermediate components providing system-wide services.

- Known upper bound on participating nodes: Fixed infrastructure networks have a known maximum number of nodes connected to the physical medium [9]. In contrast, ad hoc wireless networks have the potential to serve as a ubiquitous wireless network capable of interconnecting many thousands of devices [11]. There is potentially no upper bound on the number of nodes participating in an ad hoc wireless network. Scalability with respect to unbounded network size becomes an issue due to the increased computational load and difficulties of propagating network updates within given time bounds [12], which increases the unpredictability of wireless connections and effects timely route and resource reservation decisions.

- Static resource requirements: Both the TAO Real-time Event Service [7] and the Real-time Event Channel Model for the CAN-Bus [9], use the assumption of a static or fixed number of participating nodes with static resource requirements [9]. The assumption of a static network is used to perform real-time scheduling off-line using a reservation-based scheme to avoid collisions by statically planning the transmission schedule. An off-line admission test checks the correctness of the reservations for timing conflicts and temporal overlap. Static scheduling implies that there is no mechanism for enforcing cooperative component behaviour that would be critical in a dynamic mobile ad hoc environment.

  Static resource reservation schemes for real-time event models assume a fixed upper bound on the number of participating nodes. As previously discussed this assumption is not applicable in a mobile ad hoc wireless domain. A dynamic resource reservation scheme is required to handle the effects of dynamic mobility. To obtain mobility independent real-time guarantees, a mobile host would need to make advance resource reservations at predicted locations they may visit during the lifetime of the communication [13]. Accurate mobility and location prediction is critical for limiting the overhead of excessive resource reservation.

The dynamic mobility of ad hoc wireless networks renders the assumptions of event-based communication for infrastructure networks inappropriate. STEAM [10], is an event-based middleware service designed for the mobile computing domain, specifically IEEE 802.11 LANs utilizing the ad hoc network model. STEAM addresses some of the fundamental issues arising for event-based communication among mobile ad hoc wireless nodes. Open issues relating to the characteristics of wireless networks and their impact on the timeliness and reliability of real-time applications are not addressed. We will discuss these particular issues in the following section.

3.2.3 IMPACT OF AD HOC WIRELESS CHARACTERISTICS

3.2.3.1 Mobility related link availability

The absence of fixed infrastructure means that nodes in an ad hoc network communicate directly with one another in a peer-to-peer fashion. In wireless ad hoc networks, the topology changes dynamically and unpredictably due to node mobility [14]. Mobile nodes constitute the communication infrastructure [14] – a node acts as both a packet router and an end host. As nodes move in and out of range of other nodes, the network topology changes dynamically. The topology changes must somehow be communicated to all other nodes as appropriate. Since topology updates throughout the network cannot happen instantaneously, the global state information may never be accurate [3]. Decisions based on inaccurate information have unpredictable consequences that may be critical for real-time event-based communication.

Unlike fixed infrastructure networks where link failures are comparatively rare events, the rate of link failure due to node mobility is the primary obstacle to routing in ad hoc networks [15]. Since the rate of link failure is directly related to node mobility, greater mobility increases both the volume of control traffic and the congestion due to traffic backlogs. Link failures may result in network partitions. These observations suggest the need to include node mobility in the path selection process [11]. We propose using a predictive architecture combining mobility prediction with partition anticipation to achieve predictive routing and resource reservation. Changes in the network topology can be predicted in advance and the impact on real-time event-based communication minimized [16].

3.2.3.2 Location-Dependent Contention

Wireless transmissions in ad hoc networks are broadcast through a shared physical communication channel. Nodes within range of an ongoing transmission must restrain from sending until the channel becomes available, to avoid collisions. Hence, nodes arbitrate spatial contentions between the transmissions in their spatial locality. Luo and Lu [17] propose a fair channel access model that ensures coordination among spatially contending transmissions. However, this work assumes that individual transmissions are decoupled from each other and that scheduling can

Wireless characteristic	**Impact on real-time event-based communication**	**Benefit of prediction**
Mobility related link availability	*1) Increased computational load and unpredictability for topological updates* *2) Higher probability of link failures and network partitions* *3) Inaccurate global information*	*Mobility prediction, partition anticipation and predictive routing and resource reservation reduce reaction time to topological change by finding new routes in advance of the failure of existing routes.*
Location-dependent contention	*1) Unpredictable/unequal channel access for event flows based on inaccurate global information* *2) Inadequate network coverage*	*Coverage estimation calculates the probability of the accuracy of a decision in the presence of disconnected nodes, therefore reducing the impact of incomplete network knowledge on channel access, routing and scheduling decisions.*
Physical environment	*Unpredictable variances in connectivity*	*Anticipate partitions in advance and proactively reroute to establish new routes in advance of old routes failing.*

be performed independently for each transmission, an assumption that cannot be made in networks comprising highly mobile nodes. Also Luo and Lu, do not address temporal or reliability constraints, which are essential for providing end-to-end delivery guarantees. For real-time event transmission each mobile node must have time-bounded access to the wireless medium with a high probability. This is the focus of the TBMAC protocol [18], which also provides time-bounded collision detection and recovery, all of which are essential for event communication within bounded temporal constraints.

The area covered by the wireless transmitters of mobile nodes typically changes over time depending on the movements of individual nodes. Kanodia et al. [19] discuss achieving priority event scheduling in the presence of incomplete information sharing. They present several scenarios where the performance of IEEE 802.11 significantly diverges from an ideal reference schedule and attribute this to asymmetric information and perceived collisions. Accurate priority scheduling to reflect event class and event deadline is essential for real-time event-based communication. Any divergence from the ideal schedule is critical and in the case of hard real-time events, may even be life-threatening. Coverage estimation techniques[20], calculate the probability that there are disconnected nodes due to lack of network coverage. Using coverage estimation we can predict the accuracy of a decision in the presence of disconnected nodes. This can help to reduce the number of incorrect decisions and communication unpredictability that is critical for real-time event-based communication.

3.2.3.3 Susceptibility To The Physical Environment

Communication between mobile nodes requires the received signal strength (RSS) to be adequate to connect to another mobile node. The RSS is continually changing due to the dynamic movement of the communicating parties and the intermediary nodes at individual hops. The RSS is also significantly affected by the terrain configuration [21] for the duration of the communication too. Terrain configuration includes: hilly or mountainous areas, wooded or forested rural areas, urban areas with multistory buildings or low-density suburban areas. The dynamic changes in RSS leads to highly unpredictable connections between mobile nodes. In [20], the variance in RSS is used to anticipate network partitions. If the future state of network topology can be predicted it is possible to perform route reconstruction proactively in a timely manner [22], and find new paths prior to the failure of existing ones.

3.2.3.4 Summary

To achieve real-time event-based communication in a mobile ad hoc wireless environment the impact of the ad hoc wireless network characteristics previously identified must be limited. We propose prediction as essential to limit these wireless ad hoc characteristics. Table 1 introduces how predictive techniques limit the impediments of the wireless ad hoc environment for real-time event-based communication.

3.2.4 PROPOSED FRAMEWORK FOR AD HOC WIRELESS REAL-TIME EVENT-BASED COMMUNICATION

To achieve real-time event-based communication in a dynamic mobile ad hoc wireless network, the unpredictability inherent in the environment must be reduced. In this section we outline a conceptual model based on prediction for timely event-based communication between mobile nodes. The components and high-level interactions among them are identified in Figure 1.

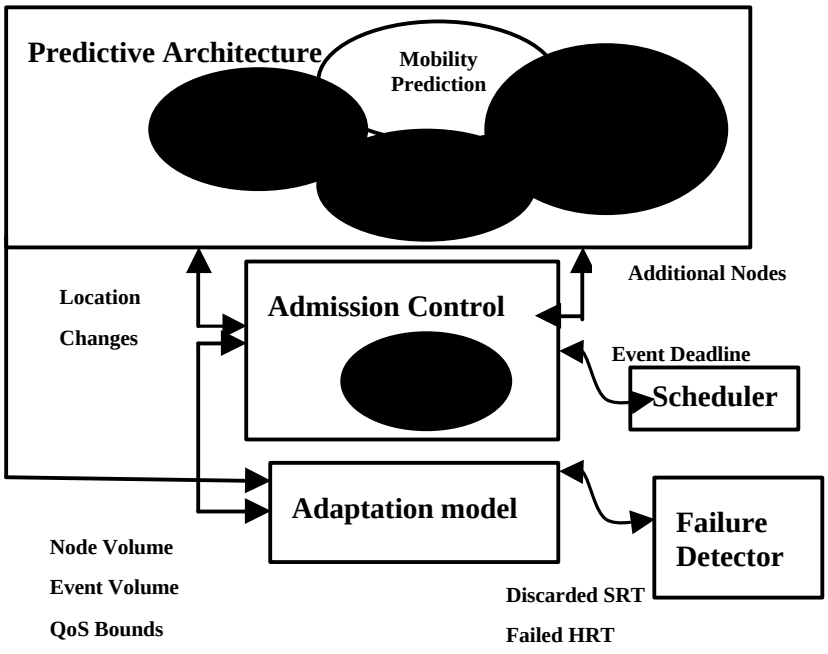


Figure 1: High-level conceptual model

Bounding the area of interest in an ad hoc network makes a large network appear smaller, but more importantly for real-time communication, it makes a highly dynamic topology appear less dynamic. Our approach is to reduce the network into dynamically organized zones, similar to proximity groups [20], which bound the number of participants, the area for maintaining topology information and the area within which event information is valid. The components of the conceptual model cooperate to maintain the timeliness and reliability requirements within a proximity-bounded zone.

The focus of our design is to reduce reaction to dynamic mobility and topological change by prediction. The admission control and adaptation components interact with the predictive architecture to make proactive decisions in advance of network change [13] .

- *Predictive Architecture:* predictive techniques are used to reduce the impact of dynamic topological changes within a zone. Location-awareness is key to determining the mobile nodes within a zone at a point in time. In our opinion location-aware routing [23] is central to achieving real-time location-bounded communication in a mobile network. We plan to extend this work to predict the future location of mobile nodes. Using this information future node movement into a zone and the impact on routing and resource reservation and timeliness and reliability guarantees for the zone can be predicted in advance.

  The ability to predict node movement contributes to achieving probabilistic guarantees of path availability due to link failure caused by node mobility. Other reasons why a link may fail, such as environment conditions or battery usage, must also be considered to avoid or anticipate network partitions. Using partition anticipation based on [20] coupled with proactive routing [24],[15] and resource reservation, we aim to improve the re-routing process by attempting to find new paths prior to the failure of existing ones.

- *Admission Control:* bounding the area of interest for real-time event-based communication implicitly limits the number of participating nodes to those within the bounded area. We apply explicit admission control policies within the zone to further reduce the number of participating nodes. The admission control policies reflect the impact of the number of participating nodes in the zone on the timeliness and reliability guarantees for a real-time event given the resources available when the real-time event is raised.

  Using predictive techniques to detect future node movement is essential for deciding the admission policy to use. For example, if resource usage is nearing maximum capacity what temporal and reliability guarantees can be made for future nodes moving into the zone and what impact does the class of real-time event have on admission control decisions?

- *Adaptation Model:* an important aspect of achieving timeliness constraints is dependable QoS adaptation [25]. However in contrast to [25] , mobility is a critical consideration. The predictive architecture detects topological changes and initiates proactive routing and resource reservation. QoS adaptation may be necessary to reflect the new routes and resources available [3]. The speed of node movement and the class of event for delivery impact the urgency of time-bounded delivery of a real-time event and impacts any QoS adaptation. Information from the predictive architecture is essential for limiting the reactive QoS adaptation required.

We propose a conceptual model to make the dynamic topology of mobile ad hoc networks less dynamic and therefore more suitable to real-time event communication. We propose that prediction is essential to reduce the reaction to dynamic node mobility and therefore essential for real-time event-based communication in wireless ad hoc networks.

3.2.5 CONCLUSION

We have outlined an approach to the complex problem of achieving real-time event communication in infrastructure-free wireless networks. We outlined the limitations of previous event-based and real-time event communication models when extended to the ad hoc wireless domain. We proposed an outline of our event-based communication model, which focuses on limiting the unpredictability of wireless communication by prediction. We described a predictive architecture for predicting node mobility, link failure and for anticipating partitions. Using this predictive architecture and QoS adaptation strategies relating to the criticality of the real-time event, we have proposed a novel approach to achieving real-time event-based communication in ad hoc wireless networks.

REFERENCES

1. A. Carzaniga "Design and Evaluation of a Wide-Area Event Notification Service". *ACM Transactions on Computer Systems*, *19* (3). 332-383.

2. G.Cugola, E. D. Nitto and A. Fuggetta "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS". *IEEE Transactions on Software Engineering, 27* (9). 827-858.

3. S. Chakrabarti and A. Mishra "QoS Issues in Ad Hoc Wireless Networks". *IEEE Communications Magazine, 39* (2). 142-148.

4. M. Gagliardi R. Rajkumar, "The Real-Time Publisher/Subscriber Inter-process communiction Model fro Distributed Real-Time Systems: Design and Implementation". in *IEEE Real-time Technology and Applications Symposium*, (May 1995).

5. Object Management Group. "CORBAservices:Common Object Services - Event Service Specification v1.1, March 2001.

6. CORBAservices:Common Object Services Specification - Notification Service Specification v1.0.1, Object Management Group, 2002.

7. A. Gokhale D. Schmidt, T. Harrison and G. Parulkar. "A High-Performance Endsystem Architecture for Real-Time CORBA" *IEEE Communications Magazine*, February 1997.

8. D. Schmidt, "ACE: an Object-Oriented Framework for Developing Ddistributed Applications". in *6th USENIX C++ Technical Conference*, (Cambridge, Massachusetts, April 1994), USENIX Association.

9. M. Mock J Kaiser, "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)". in *2nd IEEE International Symposium on Object-oriented Real-Time Distributed Computing*, (Saint-Malo, France, May 1999).

10. R. Meier and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks". in *ICDCS Workshop*, (2002).

11. R. Ramanathan and M. Steenstrup "Hierarchically-organized, multihop mobile wireless networks for quality-of-service support." *Mobile Networks and Applications, 3*.

12. B. Li, "QOS-Aware Adaptive Services in Mobile Ad-hoc Networks". in *Ninth IEEE International Workshop on Quality of Service (IWQOS 2001)*, (Karlsruhe, Germany, June 6-8 2001), ACM-Springer-Verlag, 251-268.

13. A. Talkudar, B. Badrinath and A. Acharya "MRSVP: A resource reservation protocol for an integrated services network with mobile hosts". *The Journal of Wireless Networks, 7* (1).

14. K. Wang and B. Li, "Efficient and Guaranteed Service Coverage in Partionable Mobile Ad-hoc Networks". in *IEEE INFOCOM 2002*, (New York City, New York, June 23-27 2002), 1089-1098.

15. A. B. McDonald and T. Znati "A Mobility Based Framework for Adaptive Clustering in Wireless Ad-Hoc Networks". *IEEE JSAC, 17* (8). 1466-1487.

16. R. J. Punnoose, P. V. Nikitin, J. Broch and D. D. Stancil, "Optimizing Wireless Network Protocols Using Real-time Predictive Propagation Model". in *IEEE Radio and Wireless Conference (RAWCON)*, (Denver, Colorado, USA, August , 1999).

17. Haiyun Luo and Songwu Lu. A Topology-Independent Fair Queuing Model in Ad Hoc Wireless Networks. in *Proceedings of the IEEE International Conference on Network Protocols (ICNP 2000)*, Osaka, Japan, 2000.

18. R. Cunningham and V. Cahill, "Time bounded Medium Acces Control for Ad Hoc Networks". in *POMC*, (Toulouse, France, October 30-31, 2002).

19. V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi and E. Knightly. Ordered Packet Scheduling in Wireless Ad Hoc Networks: Mechanisms and Performance Analysis. in *Proceedings of ACM MOBIHOC 2002*, Lausanne, Switzerland, 2002.

20. M. O. Killijian, R. Cunningham, R. Meier and V. Cahill, "Towards Group Communication for Mobile Participants". in *Principles of Mobile Computing (POMC'2001)*, (Newport, Rhode Island, USA, 2001), 75-82.

21. H. A. Karimi and P. Krishnamurthy, "Real-Time Routing in Mobile Networks Using GPS and GIS Techniques". in *34th Hawaii International Conference on System Sciences*, (Maui, Hawaii, January 3-6, 2001).

22. W. Su, S. Lee and M. Gerla "Mobility Prediction and Routing in Ad Hoc Networks". *International Journal of Network Management, 11* (1). 3-30.

23. Y. B. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks". in *International Conference on Mobile Computing and Networking (MobiCom'98)*, (Dallas, Texas, USA, 1998), ACM/IEEE.

24. M.R. Pearlman and Z.J. Haas "Determining the Optimal Configuration for the Zone Routing Protocol". *IEEE Journal on Selected Areas in Communication*, *17* (8). 1395-1414.

25. A. Casimiro and P. Verissimo, "Using the Timely Computing Base for Dependable QoS Adaptation". in *20th IEEE Symposium on Reliable Distributed Systems*, (New Orleans, USA, October 2001), 208-217.

Chapter 4: Quality of Service Specification

4.1 Introduction

One of the challenges that we have proposed to address in CORTEX is the need to address non-functional timeliness requirements of sentient applications in environments of uncertain synchrony. To this end, considerable work has been done around the definition of an adequate architectural solution to the problem, namely with the definition of a partially synchronous model, the Timely Computing Base (TCB) model. The adoption of the TCB model as a base for programming applications and addressing timeliness requirements has necessarily an impact on the programming model.

In deliverable WP1-D2 we have described the fundamental issues related with the problem of specifying application timeliness requirements on TCB based systems. In concrete, we studied this problem from the perspective of ensuring a certain guaranteed Quality of Service (QoS) to the application, in spite of the uncertainty of the environments. The proposed initial approach consisted in specifying timeliness QoS requirements through <bound, coverage> pairs, that is, specifying a certain time bound (relative to some timing variable of importance for the application) and an associated probability of this bound to be satisfied during the execution. Based on this approach for specifying timeliness requirements, we explained the reasoning that must be followed to satisfy such specifications. More specifically, we introduced two fundamental properties in systems of partial synchrony--- Coverage Stability and No-Contamination--- that, when secured, guarantee correct operation of applications despite timing failures. We then referred to the mechanisms underlying the programming of applications of different classes (e.g. time-safe, time-elastic), which allow the abovementioned properties to be secured for each of these classes.

In the present deliverable we extend the initial reasoning by proposing a more general way of specifying QoS requirements not limited to the observation of time bounds, based on ideas described in [1]. This extended approach is intended to address the following aspects: a) allow the integration in the same QoS framework of other application requirements, not necessarily related with timeliness requirements; b) provide the background for understanding how the several time related parameters of importance in the application context can be handled and mapped to the basic <bound, coverage> way of specifying requirements; c) provide a more flexible and configurable approach to the specification of QoS requirements, with consequent impact on the mechanisms related with adaptation.

This general framework for handling application QoS requirements, and timeliness ones in particular, does not depend on a particular interaction or communication model. The approach is sufficiently generic so that if a timing bound can be derived from some application property, then this bound will constitute the parameter of interest, the one for which a certain coverage will have to be guaranteed, in the proposed framework. However, it is important to understand how can such timing parameters be specified under specific interaction models and, in particular, when considering the event-based interaction model proposed in CORTEX. In this deliverable we overview some fundamental aspects related with the definition of what we call *generic events*, namely concerning the characterization of these events in the time domain. This is essential to allow the formulation of adequate and clear correctness criteria

from which one may derive the timing bounds that will need to be specified, observed and secured during the execution of the interactions. The presented ideas are further elaborated in [2], and will be reported in a forthcoming CORTEX deliverable dedicated to the CORTEX interaction model.

4.2 Timeliness in the CORTEX programming model: An extended model for QoS specification

In deliverable WP1-D2 we proposed a model for QoS specification in which QoS requirements were specified through <bound, coverage> pairs. We discussed the fundamental issues that must be addressed in order to build dependable applications with timeliness requirements in uncertain environments, by analysing the effects of timing failures on the application correctness. In fact, if bounds are violated during run-time, due to the occurrence of timing failures, then this may have undesirable effects on the application correctness.

The idea is to find ways of avoiding these nasty effects and keep applications correct despite timing failures. Among other things, such as the need to measure the duration of timed interactions, it is may be necessary to construct mechanisms based on *timing failure detection*, which will be used to detect the occurrence of timing failures and apply adequate fault tolerance mechanisms depending on the application.

Many applications have adaptation capabilities, which means that they may adapt their operational parameters in order to better adapt to the available resources and, in particular, to adapt to the synchrony of the environment. For example, most applications that use timeout-based communication can adapt the timeout bounds in order to avoid premature timeouts or unnecessary long waited retransmissions.

Since the approach previously proposed in WP1-D2 is focused on the specification of <bound, coverage> pairs, from the perspective of application programming this means that QoS adaptation is exclusively triggered by variations of these QoS parameters. In fact, in deliverable WP2-D3 we presented a middleware service for QoS adaptation as well as its programming interface. Note that with this approach, in which the *coverage of the bound* and not the bound itself is what counts for ensuring a certain QoS, it is possible to construct mechanisms appropriate to deal with the occasional occurrence of timing failures due to the uncertainty of the environment.

From a programming model point of view, we are talking about the different ways of adapting an application based on the output of an underlying failure detection service. Classical crash failure detection [3] is usually used for the purpose of system reconfiguration, for example, a new view in the membership of a group-oriented system. However, when: (a) the criterion for a detector to declare crash of a process or node is static; (b) the semantics of such detection is poor, then chances are that the system: (a') will not be able to adapt to the changing environment; (b') will suffer from the many mistakes of the detector.

Triggering reconfiguration in those conditions may cause instability problems in settings of uncertain timeliness. The effect on applications such as collaborative or adaptive QoS would be disastrous, with the system configuration bouncing back and forth at an unbearable rate.

We can imagine participants doing nothing but coming in and out of collaboration groups, servers joining and leaving replica sets, multimedia streams going back and forth from colour to B&W, or high to low compression. These problems still affect designs based on pure crash failure detectors, or based on rudimentary and implicit failure detection most of the times embedded in the application.

We proposed a QoS coverage service that may be viewed as a detector of failed <bound, coverage> specifications. This partially solves the problem of uncertainty, for all the cases in which it is possible to isolate a timing parameter from the overall desired level of QoS. Moreover, since the proposed approach is based on the existence timely timing failure detection, this allows for timely detection of QoS variations and fulfils an essential requirement for timely adaptation (which is of utmost importance in some applications, specially those with some criticality attributes).

Now we further extend the approach for QoS specification and QoS failure detection, based on some of the ideas presented in [1] and adapting these ideas to the partially synchronous (TCB based) framework that we follow in CORTEX. The principle underpinning the extended approach is that timing failure detectors should be *rich* and *configurable*. The combination of these properties allows dealing with application QoS requirements, not exclusively timeliness related ones, in a more flexible way and all under the same framework.

The idea is to apply the notion of a performability or quality-of-service failure detector. A QoS-FD is designed to evaluate a number of relevant operational parameters (that is, a rich semantics), in contrast with classical crash-FDs or with the basic TFD employed so far, which evaluate single parameters. For example, the QoS-FD designed for NAVTECH was oriented to assess connectivity, and thus measured [4]: roundtrip delay, throughput and omission error rate. It is important to note that some of these parameters are meaningful in the time domain, which implies that it is possible to derive timing parameters that may be observed (say, in a lower level) in order to measure them. On the other hand, some other parameters like the throughput also require the observation of non-time-related quantities, which makes the case for the usefulness of this approach when compared with the basic one. The application can configure the failure detector, by issuing a **QoS specification**. A QoS specification indicates the conditions, the goals and the importance of each parameter *P*:

Sampling Period (TS)- the interval over which the value of the parameter is acquired. If $TS = \infty$, it means continuous sampling. For any positive integer *TS*, it means the interval over which samples of *P*, or of variables leading to the computation of *P*, are collected. For example, if *P* is a round-trip time, a distribution function is computed with the samples. If *P* is a bandwidth, it is computed with the amount of bits exchanged during *TS*, divided by *TS*. If *P* is an omission rate, it means that we are interested in the number of omissions over interval *TS*. Note that this attribute will in practice force the detector to regularly produce some output, which can be understood as a form of ensuring a certain freshness of the provided QoS failure information. Although not necessary in all cases, this makes the approach more generic.

Threshold (TH)- the upper or lower acceptable bound on *P*. A typical upper bound is for a round-trip duration. A typical lower bound is for a bandwidth. A plus or minus signal is used to denote whether it is lower or upper, respectively. For example, $TH = 10^{-}$ means the parameter is valid for 10 and below (upper bound). In fact, the Threshold is

what we have simply referred to as a bound when reasoning in terms of <bound, coverage> QoS specifications. Now we generalise the definition.

Weight (WT)- a measure of the relative importance of parameter *P*. A value of 0 would mean that the parameter would not be taken into account. Comparing with the previous approach, this attribute is obviously new since we now want to deal with several parameters instead of a single one.

A test is invoked by an application, by issuing a QoS specification and a set of target resources to the failure detector. We are interested in the distribution facet of the problem, that of the QoS offered to a distributed computation through a set $\mathbb{N} = \{p_1,p_2,...,p_n\}$ of processes in different nodes that is, the end-to-end QoS seen from each p_i to every other $p_{j\neq i}$. For simplicity, in the ensuing discussion we consider one process in each node and use node and process interchangeably.

Note that our objective is to design a QoS-FD based on the existence of underlying time related services provided by a (distributed) TCB. Therefore, the resulting QoS-FD will inherit some of the properties of these underlying services, in particular those of the TCB Timing Failure Detection service. For example, given that the TCB TFD service guarantees *complete* detection of timing failures and reports the results in a consistent way at all TFD instances (in every node), it is possible to guarantee that each instance of the QoS-FD will be able to report consistent information, provided it has been collected through the TCB and that the QoS-FD is deterministic.

Each parameter of the QoS specification is evaluated by the QoS-FD at each node p_i, for every other node of the set. We note the variable *V* associated with parameter *P* kept at local node *i* about some remote node *j* as $V_i^{j(P)}$. For instance, $V_i^{j(\text{Roundtrip})}$ denotes the roundtrip delay between nodes *i* and *j* as measured at *i*. The roundtrip delay could in practice be measured using a composition of measured durations (obtained trough the TFD service, which implicitly uses the TCB duration measurement service) from node *i* to *j* and from node *j* to *i*. A test epoch is defined by the following global parameters:

QoS Spec - the specification containing the definitions for each parameter *P* (*TS, TH, WT*)

Target - the set of nodes $p_i \in \mathbb{N}$ involved.

QoS Sampling Period - the interval of observation for a QoS specification. It is possible to establish a correspondence between the QoS Sampling Period and the observation interval defined in the context of the QoS coverage service (see WP2-D3), during which observed timed actions are collected.

Value (V)- the parameter's value in the last sampling period.

Threshold Exceeded (TE)- percentage of the sampling periods of *P* where it fell beyond the bound *TH* during *TS* (or since the start, for continuous sampling), for each p_j as seen from p_i. For example, *TE* = 10 for bandwidth means that in 10% of the sampling

periods, the bandwidth was below the threshold *TH*. For example, *TE* = 2 for round-trip time in continuous sampling means that in 2% of the samples since the start the round-trip time threshold was exceeded. In other words, over time, *TE* gives a measure of the coverage of the *TH* assumption for *P*. In this last example, the round-trip time assumption holds with a probability of 98%. Just like in the previously proposed basic approach for QoS specification, this attribute is what allows the employment of techniques to achieve **coverage stability**, fundamental to the construction of dependable adaptable applications.

QoS Disturbance Index - a weighted average of the *TE* of all parameters *P*, for each p_j as seen from p_i. That is:

$$\mathrm{DI}_i^j = \sum_P \mathrm{WT}(P) * \mathrm{TE}_i^j(P) / \sum_P \mathrm{WT}(P)$$

DI Threshold - the global acceptable upper bound on the Disturbance Index.

Local Suspicion Vector (LSV) - a vector of booleans with N positions, for each local node p_i, where position $LSV_i[j] = 1$ iff the QoS Disturbance Index *DI* seen from p_i to p_j at that position exceeded (was worse than) the DI Threshold.

DI is computed periodically for each process p_i in N, as specified by the QoS sampling period. After the computation of the *DI*, *TE* variables are reset, and incremented from zero until the next computation of *DI*. A very low *DI* value means that the QoS in the current set is within satisfactory thresholds. A value of DI_i^j exceeding the *DI* threshold means a QoS failure of p_j, in the opinion of p_i. At the end of each epoch, each process p_i has thorough information provided by the QoS-FD about the QoS for the interaction with other processes, namely the $V_i^j(P)$ and DI_i^j for all *j* and all *P*. This information allows the application to fine-tune parameters, for example by choosing which ones to relax when *DI* shows insufficient QoS. Likewise, when a parameter *P* holds with 100% coverage, the application may tighten the specification. This game of tightening some and loosening others aims at the final end-to-end goal, that of obtaining the best possible QoS.

Note that depending on the specific parameters and on the means to measure them, it may be possible to ensure that all Local Suspicion Vectors are consistent with each other. Would this be the case, no other special measure would be necessary to enforce such consistency. However, when this is not “automatically” achieved, it is necessary to test each other's opinions, to assess the symmetry and transitivity of the node's view of QoS. Failure Detectors exchange their *LCV*'s to build a *Global Suspicion Matrix*, that includes all the information and all views. Some additional refinements can still be introduced, as explained in [1].

In order to adapt the application based on the information of the QoS-FD, there are several possible strategies. The actual adaptation mechanisms obviously depend on the kind of application, but in special cases, where adaptation is performed by modifying the assumed Threshold for the parameters, it would be possible to employ an extended version of the correction balance factor defined in the coverage-stabilisation algorithm presented in [5], to decide which parameter to adapt.

The actual mechanisms and protocols that would be necessary to construct the described framework are another issue that is being addressed in other CORTEX work packages, namely in the context of the system architecture.

4.3 Timeliness issues in a Generic Events Architecture

In this section we briefly describe an architecture that provides a way to structure applications around a component-based object model, allowing object composition to be influenced or constrained by the component's physical structure. The architecture postulates a generic events model easing composition and structural (body-environment) awareness, further enriching the basic CORTEX object-oriented programming model based on anonymous event-based communication.

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.

In order to address these issues, we require an event model that satisfies both functional and non-functional requirements. 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.

Here we provide just a brief description of an architecture to support such a model, which we have called the **Generic-Events Architecture (GEAR)**. This is sufficient for the purposes of this deliverable, since the focus is on the impact of non-functional (timeliness and QoS) requirements of sentient objects in the programming model, and therefore we provide a detailed discussion of the representation and handling of generic events.

The proposed architecture is depicted in Figure 1, which we describe in what follows. The L-shaped structure is crucial to ensure some of the properties described.

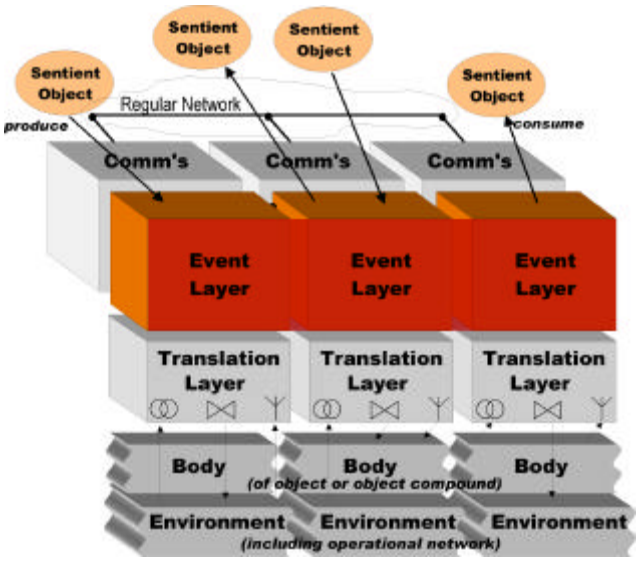


Figure 1: Generic-Events Architecture

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 for sentient objects, 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, between environment/body and an event channel. Essentially one doing observation and actuation operations on the lower side, and doing transactions of event descriptions on the other.

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 that 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 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:

- 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 Event Channels.
- 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 Channels propagate events through Regular Network infrastructures, via regular message-passing protocols executing in the Event layer.

Now, the definition of 'event' must be given more precisely. A **generic event** is a happening that takes place in the event layer at a given instant of the timeline, $<E,T^{GE}>$. The happening is internal to the system, has an event-channel compliant representation, and is not necessarily (but also) 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 timeline.

Generic events can have several origins: observation of the state or the state change of one or more real-time entities (e.g., such as produced by a smart sensor); notification about the state or the state change of one or more system variables (e.g., such as produced by a sentient object); actuation on the state of one or more real-time entities (e.g., such as produced by a sentient object for a smart actuator); action on the state of one or more system variables (e.g., such as produced by a sentient object for other sentient objects).

Further to this, event propagation is constrained to *zones*, a concept followed in CORTEX to represent the need to limit or confine the propagation of event notifications in the system. Objects can be organized into zones where a zone can be seen simply as a collection of objects and where event notifications are only propagated within the zone of the object raising the event. Objects are organized into zones at the discretion of the application programmer based on functionality, geographical location or physical location on the network. On the other hand, zones can also be seen as *QoS containers*, meaning that event channels within a certain zone are able to deliver a specified level of QoS. Thus, the scope of a given event channel flow can selectively be delimited to events of certain types (using filtering mechanisms), and the event channel modules involved in that dissemination (see Figure 1) can further be limited to those units circumscribed to a zone.

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. 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 [6]. We illustrate the nature 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 (see, for example, deliverable WP4-D8). 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}. Alternatively, a finer

synchronisation may be sought, with the *within* operator (implicitly *from* T^{GE}), so that a set-up delay of T is introduced. Even better tightness can be achieved by using at (an absolute clock time).

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: there are actions on it whose time-domain and value-domain correctness are inter-dependent. 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 [6].

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 $<r(E_i)(t_i),T_i>$ 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. Alternatively, saying, as in (4), that we know at T^{GE} that "the door is open at T", has the same constraints. Here, when observing the time at which a given discrete value E_i occurs (e.g., opening of the door), we must define the observation error (jitter) 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.

So, we must establish a bound for the errors, in order for our measurement to be useful:

- Given a known V_o, we say that an observation is **consistent** in the **value** domain, if and only if $\nu_i \leq V_o$
- Given a known T_o, we say that an observation is **consistent** in the **time** domain, if and only if $\zeta_i \leq T_o$

But this is not enough. Secondly, we must ensure that this information is sufficiently fresh to be useful. 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 [7], or *temporal accuracy interval* for control [8].

In GEAR we generalise this concept. The time of production of an event at the Event layer (T^{GE}), establishes an important *timing checkpoint* that harmonises the timing constraints of all generic events circulating in GEAR, be them concerned with real-time entities or with internal entities. The latter are "born" at T^{GE}, the former have a past history since their observation in the environment. However, note that the state of internal entities may result from previous observation and processing of sensorial information (from real-time entities) by sentient objects. The 'entering-zone mode' event in (11) is an excellent example. In that case, they

must inherit the relevant consistency constraints. Note that if t_a is the point at which actions related with those events are exerted, the above-mentioned meaningfulness is relevant both in the case of real-time entities and of internal entities. In general, this may be achieved by defining for each a maximum value error that may occur due to the passing of time. Assume bound V_s for the maximum acceptable error accumulated by a state variable S over time, since a reference instant T_i. Assume $S(T_i)$ as computed with the sensorial information available at T_i, and $S(t_a)$ as the value it would have if computed with the sensorial information available at t_a.

- Given t_a and a known V_s, we say that a state variable S is **temporally consistent at** $t_a \geq T_i$ if and only if $|S(t_a) - S(T_i)| \leq V_s$

As a particular case, temporal consistency can be secured if an interval T_s can be defined such that the variation of the value of S within that interval is at most V_s. In consequence, S would be temporally consistent *within* T_s *from* T_i, the temporal accuracy interval mentioned above.

Finally, the difference between (1,2) and (5,6) concerns the nature of the variable: called 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. In consequence, if we fabricate a 'discrete variable' which is the arrival of the crankshaft to the ignition point I, as in (9), note that this is equivalent to the kind of event in (1). This ambiguity was addressed in [6] 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*.

References

[1] P. Veríssimo and M. Raynal. Time in Distributed System Models and Algorithms. In *Recent Advances in Distributed Systems*, S. Krakowiak and S. K. Shrivastava (editors), Springer Verlag LNCS vol. 1752, 2000.

[2] P. Veríssimo and A. Casimiro. Event-Driven Support of Real-Time Sentient Objects. In *Proceedings of the 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, Guadalajara, Mexico, January 2003.

[3] T. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267. March 1996.

[4] F. Cosquer, L. Rodrigues and P. Veríssimo. Using Tailored Failure Suspectors to Support Distributed Cooperative Applications. In *Proceedings of the 7th International Conference on Parallel and Distributed Computing and Systems*, pp352-356, IASTED, October 1995.

[5] P. Veríssimo and A. Casimiro. The Timely Computing Base Model and Architecture. *IEEE Transactions on Computers - Special Section on Asynchronous Real-Time Systems*, vol. 51, n. 8, Aug 2002.

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

[7] K. Ramamritham. The origin of TCs. In *Proceedings of the 1st ACM International Workshop on Active and Real-Time Database Systems*, pp50-62, Sweden, June 1995.

[8] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.

4.4 A Description Language for the Task and Resource Models

Sentient objects are realised by both software and hardware components. Within the context of CORTEX a component is basically "a unit of composition with contractually specified interfaces and explicit dependencies only" [1]. The granularity of a component may be diverse ranging from components that realise only a part of the machinery of a single sentient object to components that encompass one or more sentient objects. The specification of both the task model and the resource model of sentient objects is achieved by expressing the resource requirements of their associated components.

An extension of the *resource configuration description language (RCDL)* presented in [2] is used for the specification of both the task model and the resource model. That is, the description language allows us to define a) the mapping between tasks and events, b) the component interfaces in which a particular task is triggered or switched, c) the system resources that are associated with each task and d) the resource requirements of a task for a particular deployment platform. The two former are achieved by the *task switch description language (TSDL)* whereas the last two are addressed by the *task description language (TDL)* and the *resource description language (RDL)*, respectively.

RCDL definitions are processed by an interpreter that is in charge of configuring the middleware according to such definitions. In particular, the code generated by the interpreter includes interceptors that are placed in the middleware to achieve task switching. Generated code also allows for the creation of the defined resource pools (VTMs) at the system initialisation time. The description languages are specified in extended BNF [3] and the complete specification is included in Appendix A. It is worth mentioning that the RCDL is a facility for programming resource requirements of sentient objects. Higher-level mechanisms are also required to map down specific sentient object requirements to specific component requirements.

4.4.1 Task Switch Description Language

There is a great variety of task configurations that can potentially exist whereby different tasks may be interconnected. For instance, a component running one task may invoke another component concerned with a different task. Such a method invocation represents a *task switching point.* Thus, a task switching point corresponds to a change in the underlying resource pool to support the execution of the task that has come into play. Task switching points are denoted as triplets including a component, an interface, and an interface operation. This approach is sufficient to specify where tasks start and where they finish.

The TSDL maps event types to tasks as shown in figure 1. Task switching points are specified on a per interface type basis and can be valid within the scope of either all components running in a node or a single component. In addition, switching points are defined for either all the operations of an interface or a single interface operation. Finally, tasks are mapped to one or more event types. "If" statements are optionally defined to determine whether a task switch should be performed according to the current task. For instance, task *f* is selected if task *g* is the current running task that invokes the method *z*.

```
TSDL description =      "Task Switching Points:",
                                {switching point}-;
switching point =               [{global task mapping}-],
                                {component}-;
component =             "component:", component name id,
                                {interface}-;
interface =             "interface:" interface type, ":",
                                [{operation}],      #if omitted, interceptor defined on all operations
                                {interface task mapping};
operation=              "operation:", operation name;
global task mapping =   "global task mapping:", task mapping;
interface task mapping = "task mapping:", task mapping;
task mapping =          {task name,":", {event type}- }- | {task name,":", event type, "if", task name}-
```

Figure 1. Specification of the TSDL

As an example consider the TSDL descriptions shown in figure 2. Two switching points are defined for the component "MyPublisher" on the operation `publish()` of the interface type `IPublish`. The task "carControl.critical" is mapped to the event type "emergencyStop". Similarly, the task "carControl.non-critical" is mapped to the event types "football", "cricket" and "weather".

```
# Publisher side
Task Switching Points:
        component: MyPublisher:
                interface: IPublish
                        operation: publish
                        task mapping:
                                carControl.critical: emergencyStop
                                carControl.non-critical : football, cricket, weather
```

4.4.2 Task Description Language

The specification of the associated resources of a task is defined by the TDL, also called *task template*. Each of the defined tasks is related to a VTM. There is a different TDL specification for each node in which the associated VTMs are defined. It should be noted that the scheduling policy of the VTMs is specified within this template as shown in figure 3. Furthermore, a task template specifies the associated tasks, the abstract resources with their related management policies, and the importance of each VTM. A high importance value is assigned to critical tasks whereas lower importance values are assigned to tasks where resource contention does not have a drastic impact in the system. In addition, sub-tasks inherit importance values from their super-tasks. There is also a *default VTM* that includes all those activities that are not represented in the resource model. That is, such activities would use the resources defined by the default VTM. The mapping of QoS values to resource parameter values may be achieved by mathematical translation or trial-and-error estimations as described in [4].

As an example of the definition of abstract processing resources, consider the particular instantiation of the resource framework whereby VTMs encompass both a team and a buffer abstraction, as shown in figure 4. The scheduling parameters of the VTM are defined which include execution time, period and CPU usage. The team abstraction is then specified in terms of a particular number of threads. Furthermore, the definition of threads include their scheduling policy along with their thread priority. The amount of buffer allocated is also defined together with its management policy. It should be noted, however, that the language is not restricted to these types of resources or the abstraction levels and can be extended to cover a different instantiation of the resource framework.

```
Task description =      "Def VTMs:",
                            "policy:", scheduling policy
                            {vtm}-,
                            default vtm,
                            {shared resources};

vtm=                    "Task:", task name,
                        "Abstract resources:", abstract resources | other hierarchy,
                        "Importance:", digit;

default vtm =           "Default VTM:",
                        "Abstract resources:" abstract resources | other hierarchy,
                        "Importance:", digit;

shared resources =  "Shared resources:", name shared resources,
                        "Abstract resources:", abstract resources | other hierarchy,
                        "Vtms:", list vtms sharing these resources;

abstract resources =    "Def VTM:",
                            "execTime:", digit,
                            "period:", digit,
                            "usage:", digit,
                            "Def Team:",
                                    "num_threads:", digit,
                                    "Def Thread:",
                                            "policy:", scheduling policy,
                                            "priority:", digit,
                            "Def Buffer:",
                                    "policy:", memory policy,
                                    "amount:",digit;
```

Figure 3. Specification of the TDL

```
Abstract resources:
        Def VTM:
                        execTime: 20
                        period: 120
                        usage: 20
                        Def Team:
                           num_threads: 5
                           Def Thread:
                              policy:
        round_robin
                    priority: 2
                        Def Buffer:
                  policy: buddy_system
                  amount: 2000
                        . . .
. . .
```

Figure 4. Example of the Specification of Abstract Resources

4.4.3 Resource Description Language

The RDL, also called the *resource template*, describes the platform-dependent resource requirements for the execution of a task instance as shown in figure 5. Whereas the TDL defines the resources provided for the execution of a task, the RDL defines the platform specific requirements for the execution of one or more interface operations. Thus, a resource template concerns the specification of aspects such as the worst-case-execution time, typical execution time, execution period, amount of memory buffer and network resources. There is also an RDL description per node as the information provided is platform specific. Thus, whenever a service is required, an admission test is performed on the basis of the resource requirements specified by the RDL. If successful, such resources are then reserved.

Resource requirements =	"**Resource Template:**", {task requirements}-;
task requirements =	"**Task**", task name, "**:**", {specific requirement}-;
specific requirement =	[cpu], [buffer], [other resource type];
cpu =	"**CPU":**", "worse case time:", digit, "typical time:", digit, "period:", digit;
buffer =	"**Buffer:**", digit;

Figure 5. Specification of the RDL

As an example of RDL descriptions, consider the specification of the resource requirements for the execution of an instance of the task "carControl.receiveNews" as depicted in figure 6. This task demands 10 ms of CPU time in the worse case whereas the normal operation of the task requires only 5 ms. The task also requires to be executed every 120 ms and demands 200 KB of memory for the purposes of buffering (again, the specification of resource requirements is not restricted to these two types of resources, thus, other type of resources may be included such as network bandwidth, storage resources and battery life).

```
Resources Template:
        Task carControl.receiveNews:
                        CPU:
                           worse case time:
        10
                           typical time: 5
                           Period: 120
                        Buffer: 200
                        . . .
```

Figure 6. Example of RDL Descriptions

References

[1] Szyperski, C. "Component Software: Beyond Object-Oriented Programming." Harlow, England, Addison-Wesley. 1998.

[2] Duran-Limon, H. A. and G. S. Blair. "QoS Management Specification Support for Multimedia Middleware." *Accepted for publication in The Journal of Systems and Software*: 2003.

[3] ISO/IEC. "ISO/IEC 14977 Extended BNF(draft).". 1996.

[4] Nahrstedt, K., H.-h. Chu, et al. "QoS-Aware Resource Management for Distributed Multimedia Applications." *Journal of High-Speed Networking, Special Issue on Multimedia Networking* 7: 227-255, 1998.

Chapter 5: APPENDIX

5.1 Event channel classes

Three event channel classes are distinguished : hard real-time event channels (HRTEC), soft real-time event channels (SRTEC) and non real-time event channels (NRTEC). The event channel classes support distinct application requirements and abstract an heterogeneous communication infrastructure. A HRTEC offers rigorous guarantees for discrete control based on sporadic events as well as for continuous control requiring periodic events like sensor readings and control feedback. For sporadic events a maximum latency will be guaranteed while for periodic events the goal is to achieve a low period- and latency-jitter. The guarantees are maintained under an anticipated number of network omission failures. Events published to a SRTEC are scheduled according to the earliest deadline first (EDF) algorithm. As outlined below, deadlines may be missed in situation of transient overload or due to the arbitrary arrival times of messages. Finally, a NRTEC disseminates events that have no timeliness requirements.

The transport of events through a HRTEC is synchronous and reliable. The properties of a HRTEC are defined by: 1.) a known upper bound for the transport latency, i.e. the interval between the point in time when an event message becomes ready and its delivery; 2.) a known upper bound for the latency jitter, i.e. the variance of the transport latency; 3.) a known upper bound of the period jitter for periodic events, i.e. the variance on the period; 4.) a fault assumption under which the properties 1.) - 3.) are valid. In order to offer such properties, a HRTEC transparently handles redundant transmissions of events and guarantees a privileged access to the communication network. Access is based on the reservation of network resources according to a TDMA mechanism (TDMA: Time Division Multiple Access) similar to the time-triggered protocol [KOP92]. However, in contrast to most TDMA schemes, the reserved time slots that are not contended by the respective HRTEC can be effectively used by weaker event channels [KAI03]. Such a flexible and efficient utilization of network bandwidth is implemented by exploiting the priority-based arbitration mechanism of the CAN-Bus.

The SRTECs support the transport of events whose temporal properties are expressed in terms of deadlines and validity intervals (expiration time). Different from HRTECs, SRTECs do not rely on reservations. Soft real-time event messages become ready at any time and are scheduled according to the earliest deadline first (EDF) algorithm. The transmission deadline is defined as the latest point in time when a message has to be transmitted. However, because a message can not be interrupted during its transmission and messages may become ready at arbitrary points in time, EDF will not always take the right scheduling decisions (only a clairvoyant scheduler would be able to do so) and situations of temporal conflicts and transient overload may occur. In theses situations, messages will still be transmitted at a later time in a best effort manner. An SRT event message eventually will be discarded if its transmission time is delayed beyond its temporal validity specified by the expiration time. The expiration time is an application specific parameter, which may be defined according to some value function.

NRTECs are used for events that do not have timeliness requirements. They are primarily intended for configuration and maintenance purposes. While HRTEC and SRTEC

disseminate events of restricted length to meet the responsiveness requirements of real-time systems, NRTEC may transfer bulk data in a sequence of message fragments.

5.1.1 Hard real-time event channels

The API for a HRTEC is presented in Figure 1. A HRTEC must configure the infrastructure according to a static schedule. An application initiates this process by calling the method: *channel.announce(subject, attribute_list, exception_handler)*.

The *announce()* method enables the local middleware components to set up the data structures representing the respective event channel and to perform the binding of the event channel subject to a network address. Three arguments are specified for the method: (i) the subject, represented by the unique identifier of the event channel, (ii) the *attribute_list*, and (iii) an exception handler. The *attribute_list* describes the specific attributes of the channel, e.g. omission degree and transmission deadline. This information is used to allocate and reserve the respective resources. When the HRTEC is configured, the application can publish events to the channel using the method: *channel.publish(event)*.

For a hard real-time channel it is not common to provide exception handling because it is based on fault masking and worst-case assumptions about temporal properties. However, it should be noted that in a distributed system, local exception handling may contribute to an early detection of a fault and thus may increase the safety of the system. The lower levels of the communication system may detect a failure, which cannot be handled by the fault masking mechanism, and propagate this information through the middleware to the respective subscribers of a channel.

```
class hrtec {

private:
subject subject_uid;

public:
// constructor and destructor of the class
hrtec(void);
~hrtec(void);

// methods used for publishing
int announce(subject, attribute_list, exception_handler);
int publish(event);

// methods used for subscribing
int subscribe(subject, attribute_list, event_queue, not_handler, exception_handler);
int cancelSubscription(void);
}
```

Figure 2. Declaration of a HRTEC class in C++.

On subscribers side, the *subscribe()* method establishes the necessary channel data structures and creates the binding of the subject to a network address. It corresponds to the *announce()* method for publishers. The *attribute_list* specifies a list of attributes used for allocating the respective resources and for filtering. For instance, we generally assume that publishers and subscribers are connected by a channel which spans multiple networks, e.g. a field bus, a wireless network and a wired wide area network. An example is described in [KAI02]. In such a scenario, a subscriber may be interested in receiving events only from publishers in the same network, i.e. those connected to the same field bus. In such a case, the respective attribute can be set accordingly and any event, which has been generated outside the field bus, will be filtered out and will not trigger a local event notification. It should be noted, however, that the HRT-channels are statically assigned to time-slots and have predefined temporal and reliability attributes. These information can be exploited in order to filter events, because only a particular type of event is allowed to be published in a certain time-slot. The known time of transmission itself therefore will be exploited as a filter for a HRT-channel.

Because events can be sporadic, the event notification service of the middleware provides an asynchronous notification mechanism for applications. When an event has passed the filters, the middleware stores the event in some predefined memory area and calls the application's notification handler. The notification handler retrieves the event from memory using the *getEvent()* primitive and then performs the required operations. As for the publisher of a HRTEC, an exception handler is also specified for the subscriber. Because a HRTEC is based on reservations, the time when a message is expected is known and thus, the event channel handler on the subscriber side can detect a missing message, rising an exception in such a case.

Finally, the *cancelSubscription()* method removes a subscription. Canceling a subscription is a strictly local operation and releases the resources in the local event handler. Only subscribers can dynamically cancel subscription to a HRTEC.

5.1.2 Soft real-time event channels

SRTECs do not use reservations. In SRTECs transmission deadlines are used to dynamically schedule the event traffic. Figure 2 depicts the declaration of the SRTEC class. Although the structure looks similar to the HRTEC, the differences are substantial and primarily are substantiated in the different attributes defined for SRTECs. Events published to a SRTEC specify a transmission deadline and an expiration parameter in the attribute list of the event. Events are scheduled by the EDF algorithm which may lead to missed deadlines because of the non-preemptive nature of the message transmission and because of transient overloads. This situation requires notifying the application for awareness reasons. Two exceptional situations may occur: a missed deadline and an expired validity. In both cases, the local exception handler is called. This local notification allows the application to react and adapt to such situations. When the validity interval is expired, the event is completely removed from the local send queue

```
class srtec {

private:
subject subject_uid;

public:
// constructor and destructor of the class
srtec(void);
~srtec(void);

// methods used for publishing
int announce(subject, attribute_list, exception_handler);
int cancelPublication();
int publish(event);

// methods used for subscribing
int subscribe(subject, attribute_list, event_queue, not_handler, exception_handler);
int cancelSubscription(void);
}
```

Figure 2. Declaration of a SRTEC class in C++.

5.1.3 Non real-time event channels

Non real-time event channels are used for events that do not have timeliness requirements. A NRTEC has a fixed priority. The priority is specified by the application during the announcement of the channel (see fig. 3). However, only priorities within a predefined range are accepted by the middleware.

NRT-channels are particularly used to configure and maintain the smart networked devices of the system. This may require to send a considerable amount of data over the network, like memory images, electronic data sheets, or test patterns. Because message frames on the CAN-Bus are limited to a payload of 8 data bytes, a mechanism to chain individual CAN messages to a larger application specific message is needed. Such a "fragmentation" mechanism for NRT channels which publishes long event messages in multiple fragments is provided by the middleware. Fragmentation is an inherent attribute of a NRT-channel and therefore, on the publisher side, fragmentation is defined during the announcement of the event channel as an entry in the *attribute_list*.

```
class nrtec {

private:
subject subject_uid;
fixed_priority fixedPriority;
boolean fragmentation;

public:
// constructor and destructor of the class
nrtec(void);
~nrtec(void);

// methods used for publishing
int announce(subject, attribute_list, fixed_priority);
int cancelPublication();
int publish(event);

// methods used for subscribing
int subscribe(subject, attribute_list, event_queue, not_handler, exception_handler);
int cancelSubscription(void);
}
```

Figure 3. Declaration of a NRTEC class in C++.

REFERENCES

[KOP92] H. Kopetz and G. Grünsteidl, "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", *Research Report No. 12/92, Inst. für Techn. Informatik, Techn. University of Vienna, 1992*

[KAI02] J. Kaiser, C. Brudna: "A Publisher/Subscriber Architecture Supporting Interoperability of CAN-Bus and the Internet", Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'2002), Västeras, Sweden, 2002.

[KAI03] Kaiser, C. Brudna, and C. Mitidieri, "A Real-Time Event Channel Model for the CAN Bus," in Proceedings of the Eleventh International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2003). Nice, France, 2003.