Detection of Out-of-Norm Behaviors in Event-Triggered Virtual Networks

R. Obermaisser, P. Peti
Vienna University of Technology, Austria

Abstract— **The DECOS system architecture integrates time-triggered and event-triggered control for combining the benefits of both paradigms. For applications based on event-triggered control, this architecture establishes event-triggered virtual networks as overlay networks on top of an underlying time-triggered physical network. In the scope of this paper, we show that the underlying time-triggered network significantly improves the accuracy of the error detection mechanisms in comparison to event-triggered architectures used today (e.g., based on Controller Area Network in the automotive industry). We introduce a framework with detectors for out-of-norm behavior, which are distributed across the nodes of the distributed real-time system. The detectors produce diagnostic messages augmented with information about the location and time of the detection event. Due to the fault isolation and the global time base of the underlying time-triggered network, the additional spatial and temporal information (besides the value domain) is available in the diagnostic messages and forms a meaningful input to a subsequent analysis process. The proposed framework manages the inherently imprecise temporal specifications of event-triggered application subsystems by correlating diagnostic messages along value, space and time. Thereby, a discrimination between a correct behavior of the computer system (e.g., triggered by rare conditions in the environment) and different fault classes (e.g., design faults, physical faults) becomes feasible. Also, the paper describes an implementation of the framework based on the Time-Triggered Protocol and explains the specification of the detectors and the analysis process using timed automata.**

I. Introduction

Advances in computer and communication technologies have made it feasible to extend the application of embedded computer systems to more and more safety-critical applications, e.g., in the automotive and avionic domain. Due to the many different and, partially, contradicting requirements, there exists no single model for building the communication system of a distributed computer system that interacts with a physical environment. Well-known tradeoffs are predictability versus flexibility, and resource adequacy versus best-effort strategies [1]. Thus, the chosen system model depends significantly on the requirements of the application.

It has been recognized that communication systems fall into two general categories with corresponding strengths and deficiencies: time-triggered and event-triggered control. Event-triggered protocols (e.g., CAN [2]) offer flexibility and resource efficiency, while time-triggered protocols (e.g., TTP [3], FlexRay [4]) excel with respect to predictability, composability, error detection and error containment. In [5] an integrated architecture (called DECOS architecture) has been introduced that brings together the benefits of both paradigms. This architecture realizes event-triggered virtual networks as overlay networks on top of an underlying time-triggered physical network. Through the support for both time-triggered and event-triggered communication activities, the integrated architecture is suitable for building mixed-criticality systems and the reuse of legacy applications. For time-triggered applications, the DECOS architecture exploits the knowledge concerning the predetermined points in time of the periodic message transmissions for error detection and the establishment of membership information [6]. These error detection services are not only employed for realizing the architectural fault isolation and fault tolerance services, but also provided as feedback to applications in order to control application level fault-tolerance.

This paper goes beyond these error detection mechanisms that are restricted to time-triggered applications only. As part of the integrated DECOS architecture, we present a solution for improved error detection services of event-triggered application subsystems. The challenge for detecting errors in event-triggered application subsystems is the inherent impreciseness of temporal specifications in this control paradigm [7]. In general, event-triggered control is chosen due to its flexibility. Since the points in time of communication activities need not be fixed at design time, the temporal behavior of communication activities can be determined on-demand by the application software. Thus, modifications of application software do not necessarily require a reconfiguration to the communication system and communication resources are only used when events occur. However, this benefit of flexibility has adverse implications for the ability of error detection [1, p. 164]. Error detection is based either on redundant computations or an a priori knowledge, while flexibility involves the limiting of a priori knowledge.

To overcome this difficulty of limited a priori knowledge, we provide a solution to error detection for event-triggered application subsystems by gathering and correlating information about improbable behavior denoted as *out-of-norm behavior*. Out-of-norm behavior is detected through the execution of deterministic timed automata, which access the consistent distributed state of the DECOS architecture as defined with respect to a global sparse time base [8].

The timed automata generate diagnostic messages as indications of out-of-norm behavior augmented with information concerning space, time, and value. The diagnostic messages are sent to an analysis process that concentrates information about multiple out-of-norm behavior occurrences in order to conclude whether an error has occurred. By augmenting each detection event with information about the locality and the global point in time of the detected out-of-norm behavior, the correlation process is significantly simplified. In particular, the meaningfulness of the spatial information is ensured through the fault isolation of the DECOS architecture [9]. The architecture's fault isolation services prevent the propagation of faults, thus allowing to pinpoint the exact location of a fault. Also, the clock synchronization service of the DECOS architecture is the basis for correlating the timestamps assigned to different out-of-norm behavior detections.

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS (Dependable Embedded Components and Systems) under project No. IST-511764.

The solution for error detection, which is proposed in this paper, is a generic architectural service and can be parameterized to adapt it to specific applications. Since control on the sending and receiving instants is under the sphere of control of the event-triggered application and not the communication system, the detectors for out-of-norm behavior need to be parameterized according to the communication model behind the application. This allows not only to detect deviations from the specification, but also to gather facts whether the underlying assumptions behind the communication model hold in reality. For instance, unanticipated customer behavior (e.g., playing with the window lifter button) may impose serious problems to the functionality of the system.

This paper is structured as follows. Section II gives an overview of the DECOS architecture. As part of this architecture, the detection of out-of-norm behavior will be discussed in Section III. Section IV outlines possible applications of the proposed solution. An implementation of the framework is the focus of Section V. The paper finishes with a discussion in Section VI.

II. DECOS Architecture

The DECOS architecture [5] offers a framework for the development of distributed embedded real-time systems integrating multiple application subsystems with different levels of criticality and different requirements concerning the underlying platform.

Structuring rules guide the designer in the decomposition of the overall system both at a functional level and for the transformation to the physical level. The services of the real-time computer system are divided into a set of nearly-independent Distributed Application Subsystems (DASs). Each DAS is further decomposed into smaller units called *jobs*. A job is the basic unit of work and exploits a *virtual network* [9] in order to exchange messages with other jobs and work towards a common goal. A *virtual network* is the encapsulated communication system of a DAS. All communication activities of a virtual network are private to the DAS, i.e., transmissions and receptions of messages can only occur by jobs of the DAS unless a message is explicitly exported or imported by a gateway. Furthermore, a virtual network exhibits predefined temporal properties that are independent from other virtual networks.

A *port* is the access point between a job and the virtual network of the DAS the job belongs to. Depending on the data direction, one can distinguish input ports and output ports. In addition, we classify ports into state ports and event ports depending on the information semantics [1] of send or received message.

In addition, the DECOS integrated architecture aims at offering to system designers generic architectural services, which provide a validated stable baseline for the development of applications (see Figure 1). The architectural services consist of *core services* and *high-level services*. The core services include a predictable time-triggered message transport, fault tolerant clock synchronization, and strong fault isolation. Any architecture that provides these core services can be used as a base architecture [10] for the DECOS integrated distributed architecture. An example of a suitable base architecture is the Time-Triggered Architecture (TTA) [11]. On top of the core services, the DECOS integrated architecture realizes high-level architectural services, which are DAS-specific and constitute

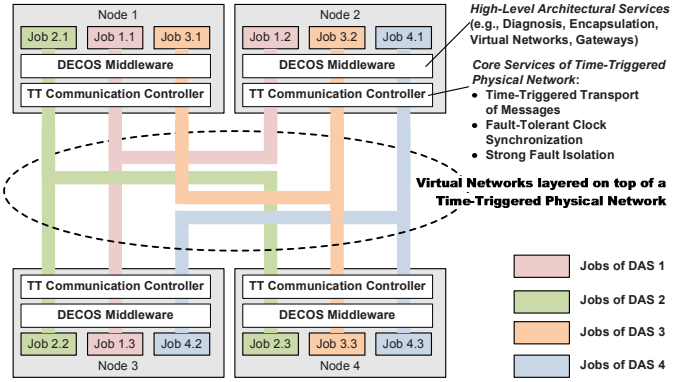

Fig. 1. DECOS Integrated System Architecture with Virtual Networks

the interface for the jobs to the underlying platform. Among the high-level services are gateway services, virtual network services, encapsulation services, and error detection services. On top of the time-triggered physical network, different kinds of virtual networks are established and each type of virtual network can exhibit multiple instantiations (see Figure 1). The encapsulation services control the visibility of exchanged messages and ensure spatial and temporal partitioning for virtual networks in order to obtain error containment.

III. Detection of Out-of-Norm Behavior

This section describes the mechanisms for the detection of out-of-norm behavior, which are part of the high-level architectural service for diagnosis in the DECOS architecture. After refining the notion of out-of-norm behavior, we introduce so-called out-of-norm detectors that are associated with the jobs. Finally, the section discusses the specification and execution of the out-of-norm detectors using timed automata.

A. Out-of-Norm Behavior

Extending the notion of *behavior* from [12], we denote a job's *behavior* as the *sequence of send and receive operations*. The behavior of a job is denoted as correct, if it is in accordance with the job's interface specification. Otherwise we speak of a job failure (i.e., a behavior violating the interface specification) and denote the respective job as faulty. In case of imprecise temporal interface specifications, however, such a demarcation between correct and faulty behavior proves to be difficult. While in a time-triggered communication system the precise temporal interface specification with a priori knowledge about the global points in time of message exchanges allows to definitely distinguish between correct and faulty temporal behavior, the imprecise interface specification of an event-triggered system complicates failure detection. When the temporal behavior of a job is determined by its inputs from the environment, an underlying model of the environment is necessary to evaluate correct job behaviors. For example, consider a user interface job in an automotive application that sends a message to a window lifter job whenever certain buttons are pressed. The ability to detect a faulty user interface job requires assumptions about the frequency and timing for pressing the button. Repeatedly sent messages within a short interval of time might represent a failure of the user interface job, or simply constitute an unanticipated customer behavior (e.g., playing with the window lifter button).

Although an omniscient observer can always demarcate between correct and faulty behavior, from within the system

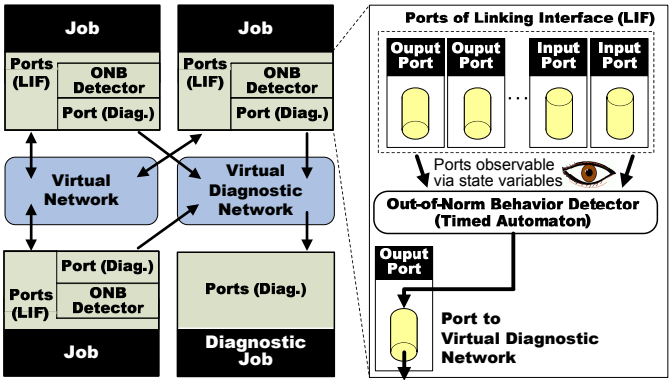

Fig. 2. Out-of-norm Behavior (ONB) Detectors based on Timed Automata

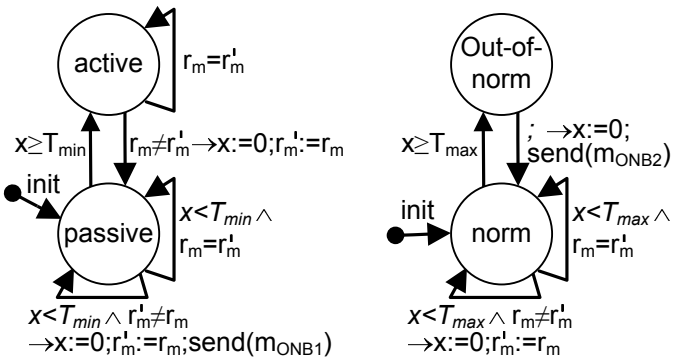

Fig. 3. Example Automaton for Detection of Out-of-Norm Behavior

the available redundancy and a priori knowledge constrain the ability for performing a definitive classification. In order to handle imprecise temporal interface specifications with limited a priori knowledge, we introduce the notion of *out-of-norm behavior*. Behavior is denoted as out-of-norm, if it cannot be classified definitively as correct or faulty at the point in time of occurrence. Out-of-norm behavior represents an improbable behavior that probabilistically represents a failure.

The notion of *out-of-norm* originally comes from production machinery where early warnings of system breakdown enable preventative maintenance, thereby achieving enormous cost savings [13]. In this paper we extend this concept to the domain of computer systems by providing a generic mechanism that allows to detect suspicious behaviors in the temporal and value domain.

B. Out-of-Norm Behavior Detector

At each job, a so-called *out-of-norm behavior* detector monitors the behavior at the ports to the virtual network. When out-of-norm behavior is detected, a corresponding diagnostic message is disseminated via a so-called *virtual diagnostic network*. This virtual diagnostic network is established by exploiting the high-level virtual network service. Such a virtual solution has two main advantages. At first, real-time traffic is not compromised in any way since the bandwidth for the exchange of diagnostic information is fixed a priori at design time. This way a deterministic message exchange for all non-diagnostic DASs is guaranteed. Secondly, the purely virtual solution ensures that no additional hardware faults are introduced due to wiring or connector problems. Consequently, *no probe effect* can be introduced [14]. By exploiting the architectural high-level services to establish the virtual diagnostic network, no back-propagation of the diagnostic dissemination service to safety-critical subsystems is possible, since only elementary interfaces are used [15].

Consequently, at least two ports are associated with every job. One port, which is an output port, is a connection to the virtual diagnostic network (see Figure 2). The other ports form the *linking interface* [16] of the job, via which the job is connected to the other jobs of the DAS. These latter ports are the access points to the DAS's virtual network, which serves the exchange of messages between jobs to realize the emergent services.

Whenever the out-of-norm behavior detector recognizes out-of-norm behavior at the ports of the linking interface, the detector sends a diagnostic message at the virtual diagnostic network with the following information:

- **Time:** The message sent to the virtual diagnostic network contains a timestamp denoting the point in time of the out-of-norm behavior detection.
- **Space:** The location (job and DAS) where the out-of-norm behavior has been detected.
- **Value:** The type of the out-of-norm behavior (e.g., with respect to the dynamics of the real-time entities, such as an improbable gradient in the message values) and the state variables that are significant for analyzing the out-of-norm behavior (i.e., contextual information).

C. Specification of Out-of-Norm Behavior Detectors

Each out-of-norm behavior detector is specified using a deterministic timed automaton, i.e., a state transition graph annotated with timing constraints. We extend the timed automata defined in [17] by supporting not only clock variables, but also state variables with finite value domains. The state variables can be declared by the user within a timed automaton. In addition, predefined state variables are used to monitor the ports of the linking interface. These predefined state variables include:

- **State variables for messages.** A port to an event-triggered virtual network is a queue. Each position in this queue is made accessible to the out-of-norm behavior detector via a corresponding state variable. The access to this state variable does not affect the data at the port, i.e., no messages can be modified or retrieved from the queue.
- **State variables for port status.** These state variables contain information about the ports, such as the number of queued messages, and the total number of sent or received messages.

Figure 3 contains exemplary timed automata for monitoring the interarrival time of incoming messages. The automaton on the left-hand side of Figure 3 contains two states (active and passive) for detecting violations of the minimum interarrival time T_{min}. The actual interarrival time of the message is captured with the clock variable x. The detection of the arrival of a message occurs using the predefined state variable r_m denoting the number of received messages. r_m is compared to a user-defined variable r'_m with the previous value of r_m in order to react to an increase of the number of received messages. In the active state, x is larger than T_{min} and a new message may arrive. In the passive state, the minimum interarrival time has not yet elapsed and the arrival of a message represents an out-of-norm behavior. In this case, a diagnostic messages is sent ($send(m_{\text{ONB1}})$) via the virtual

diagnostic network. The information concerning the spatial and temporal context is automatically added.

The timed automaton on the right-hand side of Figure 3 detects violations of the maximum message interarrival time $T_{\max}$. This example is of particular importance for today's automotive applications, because many automotive ECUs provide a life sign with a maximum message interarrival time. In case the application in an ECU does not send a message with user data during an interval of length $T_{\max}$ (e.g., park assist above 30km/h), a life sign message is produced controlled by a timeout. Like in the previous example, the clock variable x captures the actual interarrival time of the message. x is reset whenever a messages arrives. In case x exceeds $T_{\max}$, an out-of-norm behavior has occurred and a diagnostic messages is sent ($send(m_{\text{ONB2}})$).

D. Execution of Out-of-Norm Behavior Detectors

The execution of the timed automata is controlled by the progression of time on a global sparse time base [8]. In the sparse time model the continuum of time is partitioned into an infinite sequence of alternating durations of activity and silence. Thereby, the occurrence of significant events is restricted to the activity intervals of a globally synchronized action lattice. The sparse time base allows to generate a consistent temporal order on the basis of time-stamps [1]. During the silence intervals of the action lattice the sparse time base provides a consistent distributed state, where the notation of state is used as introduced in system theory [18, p. 45] as a dividing line between past and future.

Every port of a job is associated with a corresponding port state, which is the state of the job as seen from the port. The notation of *port state* is based on the concept of interface state [12]. For a port with state semantics, the port state comprises a state variable, while a message queue is the port state of a port with event semantics. The update of the port state through the virtual network service occurs during the activity intervals of the global sparse time base. A message reception from a virtual network results in the update the state variable associated with a input port with state semantics or the insertion of an event message into a message queue of an input port with event semantics. At an output port with event semantics, message transmissions via a virtual network results in the removal of messages from an outgoing message queue. In the silence intervals of the sparse time base, globally consistent input is provided via the port state to the diagnostic services (see Figure 4).

In these silence intervals of the virtual network service, the out-of-norm behavior detector accesses the ports via the execution of the deterministic timed automata. Thereby, each automaton can start its execution with a globally consistent port state. At each activation, the automata associated with the job are executed and the labels associated with the transitions of the automata cause the observation of ports and the production of diagnostic messages.

Upon each activation time progresses by n ticks, which is reflected by all clock variables maintained in the timed automata. For each automaton, the out-of-norm behavior detector proceeds with evaluating guards and performing state changes with edges for which all guards are being fulfilled. In case no edge can be taken with the current values of the variables (including clock variables), the execution of the respective automaton is finished for this activation cycle. During the execution of the automata, the reading of the port state is triggered through the execution of transitions accessing the predefined state variables. The taking of an edge with a label $send(...)$ allows to transmit a diagnostic message.

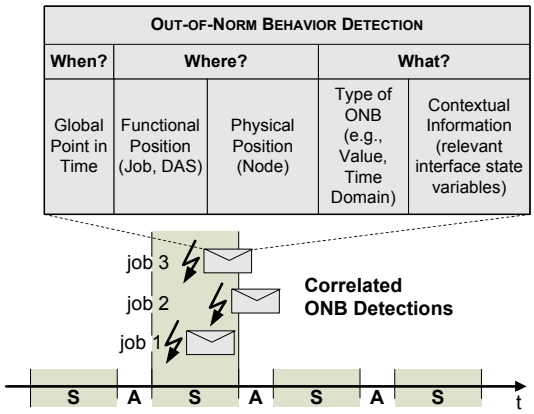

Fig. 4. Diagnostic Messages. *The architecture provides indications of out-of-norm behavior at the action lattice of a global sparse time base with corresponding activity (A) and silence (S) intervals.*

During the activity interval of the virtual network service, on the other hand, the out-of-norm behavior detector is inactive in order to prevent concurrent access of the port state between the timed automata and virtual network service.

IV. Exploitation of Out-of-Norm Behavior Detections

Once *out-of-norm behavior* of a job is detected, information about the occurrence of this behavior is disseminated so it can be used as input to other services at the architecture or application level:

1) **Maintenance services** process information about the occurrence of out-of-norm behavior to determine whether jobs or nodes need to be replaced.
2) **Engineering feedback services** process information about out-of-norm behavior in order to generate engineering feedback (e.g., performance monitoring, information regarding the bandwidth utilization).
3) **Membership services** are architectural services that solve the membership problem [19] by achieving agreement on the identity of all correctly functioning jobs of a DAS. A membership service simplifies the provision of many application algorithms and plays an important role for controlling application level fault-tolerance mechanisms.

The key element to all above mentioned applications is the inclusion of meaningful information about the *space*, *time*, and *value* of an occurrence of out-of-norm behavior indicated through a diagnostic message. The information in a diagnostic message is trustworthy in the sense that this information has been acquired from independent instances, i.e., from architectural services executes at different jobs and at different nodes. Based on the fault hypothesis of the DECOS architecture [5], nodes are independent Fault Containment Regions (FCRs) with respect to hardware faults and jobs are independent FCRs for software faults. We denote the underlying design concept as *cross checking* [20], [21], i.e., different FCRs are evaluating checks upon each other.

In the context of *space*, error containment between virtual networks [22] ensures that the detected out-of-norm

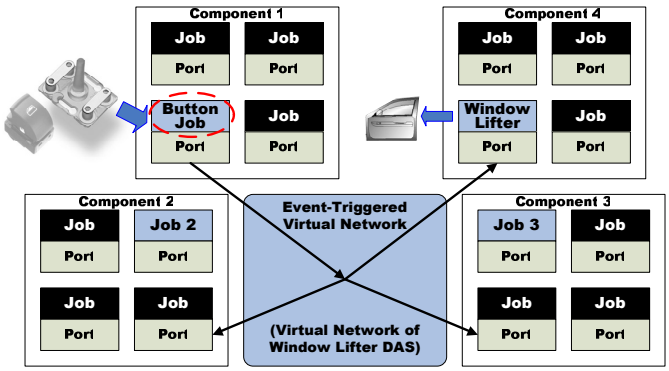

Fig. 5. Exploitation of Out-of-Norm Behavior Detections

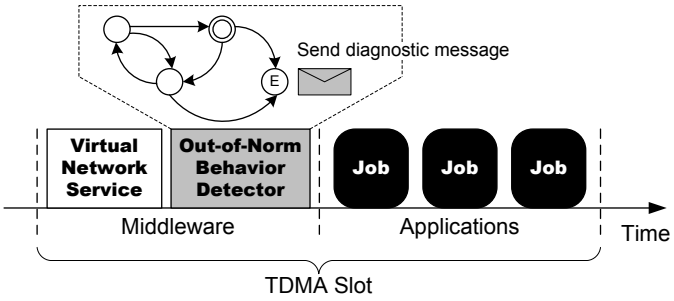

Fig. 6. Overview on the Software Executed in each Node

behavior enables to constrain the problem to a certain DAS. Consequently, the out-of-norm behavior cannot result from interference with another DAS). Consider, on the other hand, a CAN system [2] with two DASs. If the two DASs share a common CAN bus, then a failure in one DASs (e.g., babbling idiot failure with high priority messages) can propagate into the other DAS since the message transmissions of one DAS can delay message transmission of the other DAS,

Furthermore, the DECOS architecture guarantees that the predefined temporal properties for the communication activities hold also in case of job failures within the DAS [22]. Although an out-of-norm behavior may propagate within a DAS, the out-of-norm behavior detection services allow to trace the job representing the origin of this phenomenon as well as the spreading within the DAS. Thus, error containment between jobs permits to pinpoint the jobs that are responsible for a specific out-of-norm behavior.

Time information is meaningful due to the availability of a global notion of time in the integrated DECOS architecture. The out-of-norm behavior detection events at different jobs occur within the same silence interval of the global sparse time base. Furthermore, all jobs in a DAS have access to a consistent port state, which is a prerequisite for the correlation of out-of-norm behavior detections resulting from the cross checking between jobs.

Value information captures the type of out-of-norm behavior (e.g., improbable frequency of messages sent by a job, implausible gradient of a sensor value) as well as contextual information given by the relevant part of the port state at the point in time of the out-of-norm behavior event.

Figure 5 exemplifies the exploitation of the out-of-norm behavior detection in an automotive example. Whenever a window lifter button is pressed, a corresponding job of the window lifter DAS disseminates a message with information about this event to an actuator job that accesses the motor. Out-of-norm behavior detectors are located at all jobs of the DAS. These jobs execute deterministic timed automata accessing the port state in order to determine a possible out-of-norm behavior of the button job.

In case the button job is subject of out-of-norm behavior induced by either a software or hardware fault, the other jobs of the DAS (actuator job, and jobs 2 and 3) will collectively detected this out-of-norm behavior. The checks are being executed in the other nodes in order to realize the cross checking principle and this information is forwarded to an analysis process. The analysis process exploits this information for the construction of maintenance services, engineering feedback services, or membership services. This analysis process concentrates this information and correlates it with other out-of-norm behavior detections from different functional elements (DASs, jobs) and information from different physical elements (nodes). For maintenance purposes, correlated job failures of different DASs within a node are an indication of a hardware fault [23]. Isolated job failures are an indication for software faults. The trustworthiness of the space, time, and value information in diagnostic messages is essential for determining during the analysis process whether a hardware or software faults is affecting the state of the system.

V. Implementation

The target platform used for the implementation of the introduced concept is based on the Time-Triggered Architecture (TTA) an exploits TTP [3] as the core physical network of the integrated system. The TTA provides a computing infrastructure for the design and implementation of dependable distributed embedded systems up to the highest criticality class [11].

The nodes employed in the prototype implementation use an embedded real-time Linux variant extended by a time-triggered scheduler as their operating system [24], [21]. For the communication system, the prototype uses an implementation of multiple virtual networks on top of TTP [9]. Each TTP frame is partitioned according to the number and requirements of the jobs of the DASs hosted on the respective DECOS node. This splitting of available communication resources and the establishment of the protocol specific properties (e.g., transformation of state variables into queues as required in event-triggered communication) is performed by the middleware deployed on each DECOS node.

For the implementation of the out-of-norm behavior detection service, we have added another middleware layer in this prototype implementation of the DECOS architecture. A generic out-of-norm behavior detector is parameterized by the operational interface specifications of jobs and executed as a time-triggered task. Each job's operational interface specification comprises an XML description as defined in [25]. The XML description denotes the syntax of exchanged messages and one or more timed automata expressing different classes of out-of-norm behaviors for jobs. Out of this XML description, tools create software modules for the middleware task performing out-of-norm behavior detection. The out-of-norm behavior detector accesses the virtual network service, which is also realized as a middleware task, in order to acquire information about the job's message transmissions and receptions. This information controls the execution of the timed automata and therefore determines when diagnostic message are sent via the virtual diagnostic network.

Figure 6 illustrates the software that is executed during each TDMA slot of a node. The execution of the virtual network service ensures that the messages for each job hosted on the node are transferred using the underlying TTP network. In the out-of-norm behavior detector, the timed automata are executed in order to detect improbable behaviors and deviations in the value and time domain from the respective port specification. Upon a detection event, a diagnostic message is constructed and disseminated via the virtual diagnostic network to the analysis subsystem for further assessment in order to determine the fault. As indicated by the time line in Figure 6, as soon as the middleware (i.e., for out-of-norm behavior detector and virtual network service) has terminated, the jobs can be executed (depending on the node configuration, one or more jobs are executed in one TDMA slot).

VI. Discussion

The key for effective error detection is a precise job interface specification in the syntactic, temporal, and semantic domain. However, such sharp specifications may not always be available, especially in systems where flexibility is more important than predictability. Detection of out-of-norm behavior is especially useful, if the specification includes imprecise specifications such as probabilistic timing assumptions as in event-triggered communication systems. Typically, this information is expressed via probability distributions, thus a sharp line that allows a classification into correct and incorrect cannot be drawn. In this paper we provide a mechanism that allows to cope with this uncertainty by classifying this gray area as out-of-norm. The execution of timed automata that encode checks in the value and temporal domain allows to detect those interface states that bear the potential of revealing job faults.

Based on the error containment of the DECOS architecture provided via the underlying time-triggered core network and high-level encapsulation services, it can be assured that the detected out-of-norm behavior does not originate from unintended side effects or mutual dependencies between jobs having no functional dependencies. This ensures that out-of-norm behavior can be indisputably assigned to one or more jobs having functional dependencies. Furthermore, failure modes such as babbling idiot or masquerading failures are precluded by the architecture (i.e., virtual network service) and must not be dealt with at the application level.

While typically, executable assertions can only capture incorrect data in the value domain [26] the proposed solution can also capture out-of-norm behavior in the time domain, thus providing information about the dynamics of the system.

The introduced framework provides a solution to gather diagnostic information that can be used for either maintenance, feedback to jobs, or performance monitoring in order to judge about the effectiveness of the application functionality in the field. In case of maintenance, the information can be used as an input for online diagnosis in order to judge about the health status of the integrated system.

Acknowledgments

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

References

[1] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[2] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[3] H. Kopetz and G. Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994. Vienna University of Technology, Real-Time Systems Group.

[4] FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.

[5] R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006. Available at http://www.springerlink.com.

[6] A. Ademaj. *Assessment of Error Detection Mechanisms of the Time-Triggered Architecture Using Fault Injection*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[7] R. Obermaisser. *Event-Triggered and Time-Triggered Control Paradigms – An Integrated Architecture*. Real-Time Systems Series. Kluwer Academic Publishers, November 2004.

[8] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proc. of 12th Int. Conference on Distributed Computing Systems*, Japan, June 1992.

[9] R. Obermaisser and P. Peti. Realization of virtual networks in the DECOS integrated architecture. In *Proc. of the Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS)*. IEEE, April 2005.

[10] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, September 2001.

[11] H. Kopetz and G. Bauer. The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software*, January 2003.

[12] C. Jones et al. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585)*, December 2002.

[13] G.T. Jermy, R.C. Castle, C.F. Gimblett, and R. Chakrabarti. Monitoring out of normal conditions in repetitive cycle production machinery. In *Proc. of the Fifth Int. Conference on Factory 2000*, pages 29–33. IEEE, April 1997.

[14] J. Gait. A probe effect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.

[15] H. Kopetz. Elementary versus composite interfaces in distributed real-time systems. *ISADS '99, March 1999, Tokyo, Japan*, Mar. 1999.

[16] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. Research report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[17] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[18] M. D. Mesarovic and Y. Takahara. *Abstract Systems Theory*, chapter 3. Springer-Verlag, 1989.

[19] F. Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4:175–187, 1991.

[20] P. Peti, R. Obermaisser, and H. Kopetz. Out-of-norm assertions. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, March 2005.

[21] P. Peti. *Diagnosis and Maintenance in an Integrated Time-Triggered Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, September 2005.

[22] R. Obermaisser, P. Peti, and H. Kopetz. Virtual networks in an integrated time-triggered architecture. Research report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[23] R. Obermaisser and P. Peti. A fault hypothesis for integrated architectures. In *Proc. of the 4th Int. Workshop on Intelligent Solutions in Embedded Systems*, June 2006.

[24] B. Huber, P. Peti, R. Obermaisser, and C. El Salloum. Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems*, May 2005.

[25] R. Obermaisser and P. Peti. Specification and execution of gateways in integrated architectures. In *Proc. of the 10th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA)*, Catania, Italy, September 2005. IEEE.

[26] B.M. McMillin and L.M. Ni. Executable assertion development for the distributed parallel environment. In *Proc. of the Twelfth Int. Computer Software and Applications Conference (COMPSAC 88)*, pages 284–291. IEEE, October 1988.