A Diagnostic Framework for Integrated Time-Triggered Architectures

P. Peti and R. Obermaisser
Vienna University of Technology, Austria
email: {php,ro}@vmars.tuwien.ac.at

Abstract

Integrated architectures promise substantial technical and economic benefits in the development of distributed embedded real-time systems. In the context of diagnosis new diagnostic strategies can be applied by taking the physical and functional structure of an integrated system into account. In this paper we present a diagnostic framework that is designed to tackle prevalent diagnostic problems industry is currently facing, such as the trouble-not-identified phenomenon in electronic systems. So-called Out-of-Norm Assertions (ONAs) are employed that combine diagnostic information to correlate experienced failures in order to decide on the type fault (e.g., transient vs. permanent, internal vs. external) affecting the system. Based on a prototype implementation of the integrated time-triggered DECOS architecture we show the feasibility of this diagnostic strategy.

1 Introduction

The development of effective diagnostic systems stayed behind the recent increase of electronic systems in modern means of transportation. One reason for the diagnostic deficiencies of modern On-Board Diagnosis (OBD) systems is the fact that diagnosis is often treated as an add-on to electronic systems rather than an integral part of the architecture.

For instance, when an OBD system of a car detects a violation of the specification of an Electronic Control Unit (ECU), a breakdown log entry is written, recording the condition of the vehicle when a failure occurs. This so-called *freeze frame* provides important information for the failure cause analysis. The maintenance engineer can use this collected data for getting insight into the context of the system malfunction. However, this information is often insufficient to understand the complex processes within the system that caused the subsystem to fail, since typically only local information is provided that does not allows to correlate experienced component failures at different parts of the system [1]. As a consequence, the problem of the identification of faulty Field Replaceable Units (FRUs) has become a predominant challenge that needs to be solved in distributed embedded real-time systems.

In order to cope with industry demands on diagnosis, the *detection* and subsequent *identification* of the FRUs causing malfunction should be supported by the system architecture. In order to include diagnosis into the design process a framework should be provided that can be parameterized according to the developer's needs and thus avoids costly redesign of individual diagnostic solutions at application level. In particular, the system architecture needs to support the maintenance engineer by providing information about the health status for each FRU that acts as a foundation for the decision process whether a FRU remains in the system or will be replaced. This online analysis of the gathered diagnostic information is required for future generations of computer systems to reduce the numbers of *cannot duplicate* failures, i.e. failures that cannot be reproduced at the service station (see for instance [2]).

In this paper we look at diagnosis in the context of integrated architectures. Integrated architectures promise massive cost savings due to the multiplexing of hardware resources among different applications. Furthermore, the resulting reduction of wiring and connectors results in dependability improvements. In addition, integrated systems permit a better tactic coordination of tightly coupled control activities in different application subsystems. For this reason, integrated architectures are becoming more and more interesting for deployment in the automotive domain, in order to resolve the pending *one function – one ECU problem* associated with increasing complexity and costs.

In this paper we present the diagnostic framework of the DECOS integrated architecture that deploys Out-of-Norm Assertions (ONAs) to judge about the condition of the constituting FRUs of the integrated system. ONAs take the characteristics of faults in the time, value and space domain into account in order to discriminate between different types of faults that are affecting the operation of the distributed system. ONAs operate on the distributed state of the system in order to allow correlation of experienced failures to improve the accuracy of the analysis process. Our solution adheres to the following requirements imposed by industry:

- **Focus on Transients.** The types and causes of failures for electronics have changed over the years. Failure analysis in recent years has revealed that permanent failures have been reduced by improvements in technology but due to the higher level of complexity and downsizing other failure classes have emerged [3]. The tremendous improvements made by the IC industry

with respect to permanent failure rates are extenuated by increasing transient failure rates for instance due to semiconductor process variations, shrinking geometries, and lower power voltages [4]. Consequently, the diagnostic services must especially be designed to handle transients.

- **Detection of Correlated Errors.** Diagnostic systems operating on only the internal component state preclude the possibility to detect and analyze correlated failures or system anomalies at different nodes.

- **Avoidance of the Probe Effect.** Any diagnostic subsystem must avoid the introduction of probe effects [5] that may forge the outcome of the diagnostic subsystem. This is especially important in case of real-time systems, where the diagnostic messages must not compromise the real-time traffic in any way.

- **Intellectual Property Protection.** Diagnosis is often equated with revealing of internal information. The presented framework allows the realization of advanced diagnostic strategies by solely operating on the interface state of the linking interfaces [6] without revealing any internals of the application.

The paper is organized as follows. Section 2 introduces the DECOS integrated architecture, including an overview on the diagnostic services of the architecture. In Section 3 a framework for diagnosis is presented that can be parameterized by the system and application designers. Within this framework the deployed ONAs are executed and evaluated. In Section 4 we elaborate on the specifcation of ONAs using timed automata. Furthermore, we discuss the execution scheme that builds upon the sparse time base of the underlying time-triggered core system. In Section 5 the prototype implementation of the framework is described. On the basis of an ONA for the detection and classification of borderline failures (e.g., connector failures) we show the feasibility of this diagnostic strategy.

2 The Integrated Diagnostic Architecture

This section describes the DECOS integrated architecture for dependable distributed embedded real-time systems [7] and focuses on the integrated diagnostic services. The DECOS integrated architecture provides a framework with generic architectural services for integrating multiple application subsystems within a single, distributed computer system, while retaining the error containment and complexity management benefits of federated systems.

2.1 Functional System Structure

For the provision of application services at the controlled object interface, the real-time computer system is divided into a set of nearly-independent subsystems, each providing a part of the computer system's overall functionality. We

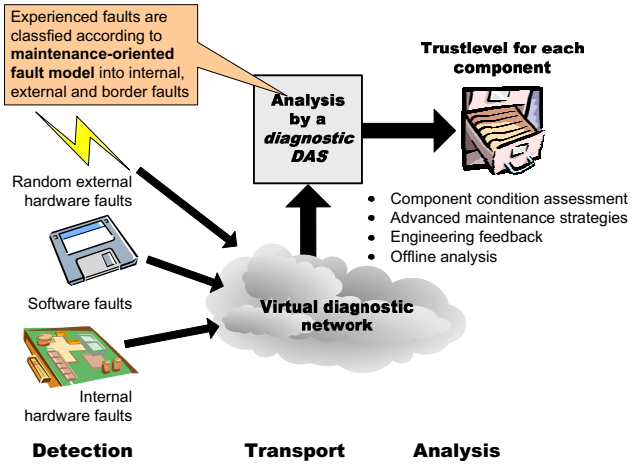

Figure 1. Diagnostic Services of DECOS

denote such a subsystem as a *Distributed Application Subsystem (DAS)*, since the implementation of the corresponding functionality will most likely involve multiple components that are interconnected by an underlying communication system. The implementation as a distributed system is a prerequisite for establishing fault-tolerance by redundantly performing computations at separate components that fail independently. In addition, the DECOS integrated architecture groups DAS with the same criticality into subsystems (e.g., safety-critical vs. non safety-critical).

In analogy to the structuring of the overall system, we further decompose each DAS into smaller units called *jobs*. A *job* is the basic unit of work that employs a *virtual network* for exchanging information with other jobs, thus working towards a collective goal. The access point of a job to the virtual network is denoted as a *port*. Depending on the data direction, one can distinguish input ports and output ports. A job employs input ports for exploiting the services of other jobs, while output ports enable a job to provide its own services. Depending on the application context, either the event-triggered or time-triggered paradigm is supported by the virtual network infrastructure.

2.2 Waist-Line Architecture

The integrated DECOS architecture is based on a time-triggered core architecture that meets the safety requirements of ultra-dependable applications. The core of such an integrated distributed architecture for time-critical systems must provide four *core services*: predictable transport of messages, fault-tolerant clock synchronization, strong fault isolation, and consistent diagnosis of failing nodes. Any architecture that provides these core services can be used as a base architecture for the DECOS integrated system architecture.

Based on the core architecture, *high-level services* such as a virtual network service as the communication infrastructure tailored to the needs of each DAS, an encap-

sulation service for ensuring inner-component error containment, hidden gateways for the interconnection of DAS to improve quality of service and eliminate resource duplication, a redundancy management service (e.g., voting), and the diagnostic service as illustrated in Figure 1 are deployed. In the remainder of this paper we will focus on the diagnostic service.

2.3 Integrated Support for Diagnosis

The model of the diagnostic architecture [8] as illustrated in Figure 1 can be divided into three consecutive steps. Once a fault (either software or hardware) affects the system and is detected by the detection mechanisms of the diagnostic services, a corresponding message is disseminated via a dedicated *virtual diagnostic network*. A virtual network is an encapsulated overlay network on top the time-triggered core physical network [9]. The high-level virtual network service ensures that fault isolation between virtual networks of different DASs is guaranteed and that temporal properties are not invalidated by interference of communication activities between different DASs. The subsequent analysis of this information is located in an encapsulated *diagnostic DAS* in order to determine the nature of an experienced fault with respect to a maintenance-oriented fault model. The diagnostic DAS outputs a trust level for each component, that acts as the foundation for the decision of the maintenance engineer on the question whether a FRU should be replaced or remain in the system.

The diagnostic framework decouples *architecture level* diagnosis from *application level* diagnosis to reduce the complexity and the associated efforts for the application developers. These mechanisms are parameterized by the application developers and eliminate the need for the deployment of proprietary solutions. The architecture level diagnosis techniques (e.g., identification of component hardware failures) are independent of a particular application and need not be covered by the respective application level diagnosis strategy (e.g., plausibility checks). In addition, a revalidation of the systematic diagnosis mechanisms by the manufacturer is rendered obsolete if the coverage of the deployed mechanisms has been validated.

2.3.1 The Maintenance-oriented Fault Model

As stated in [10] the concept of fault is introduced to stop the recursion of the "fault-error-failure" chain. From a maintenance point of view, we are only interested in categorizing the type of fault of the experienced failure into classes that allow a determination whether a replacement is the correct maintenance strategy. Thus, by traversing backwards the fault-error-failure chain [10], it must be possible for the diagnostic subsystem to determine whether a change of a FRU can eliminate the experienced problem, or if a replacement (i.e. change of hardware or update of software) will prove to be ineffective. On the basis of the maintenance-oriented fault model a corresponding maintenance action for each fault class needs to be stated.

Dimension	Fault Patterns			
	Wearout	Massive Transient	Connector Fault	Software Fault
Time	increasing frequency as time progresses	approximately at the same time (within a small delta)	arbitrary	peak-load scenario, clustered
Space	one component only	multiple components with spatial proximity	one component	components with communication relationship
Value	increasing deviation from correct value, at the verge of becoming incorrect	multiple bit flips	message omissions, syntactic invalid frames on a channel	message omissions

Figure 2. Summarized Fault Patterns

Consequently, we stop the recursion at Field Replaceable Unit (FRU) level. In the context of the DECOS architecture in case of hardware faults the FRU is considered to be system component (i.e. node computer), while for software faults the FRU is considered to be a job. The fault classification for each FRU needs to be derived by analyzing the prevalent types of faults affecting the given FRU. For a detailed discussion on the maintenance-oriented fault model see also [11].

2.3.2 Operation on the Distributed State

The pivotal strategy of the diagnostic services is the operation on the *distributed state* established via the underlying core services of the integrated architecture due to the availability of a global sparse time base. In combination with the rigorously applied encapsulation services (both at inter- and inner-component level) [9, 12], this strategy allows to trace correlated system anomalies back to the FRU responsible for the experienced system behavior.

Whenever a fault affects one or more constituting parts of the distributed system, a change of state can occur that leads to an unintended state denoted as an error [10]. Depending on the type of fault (e.g., internal or external fault, software or hardware fault), the unintended state will exhibit a characteristic manifestation in time, value and space. To capture the characteristics of the fault-induced distributed state changes, we introduce the concept of *fault pattern* [13]. A fault pattern is the set of state variables that has been identified as subject to fault-induced state changes along with corresponding properties in value, space, and time. Different types of faults show different fault patterns on the distributed state (see Figure 2 for some typical examples).

2.3.3 Out-of-Norm Assertions

In the diagnostic architecture so-called *Out-of-Norm Assertions (ONAs)* [13] are deployed that are checked against the distributed state established by the use of a sparse time base [14]. We define an *Out-of-Norm Assertion (ONA)* as a predicate on the distributed system state that encodes a *fault pattern* in the value, time and space domain. ONAs are triggered, whenever all symptoms of a particular fault pattern are detected on the distributed state. A *symptom* is a condition on a set of local interface state variables of a particular component that is monitored to detect deviations from the interface specification. An ONA will likely be composed of more than one symptom, each operating on

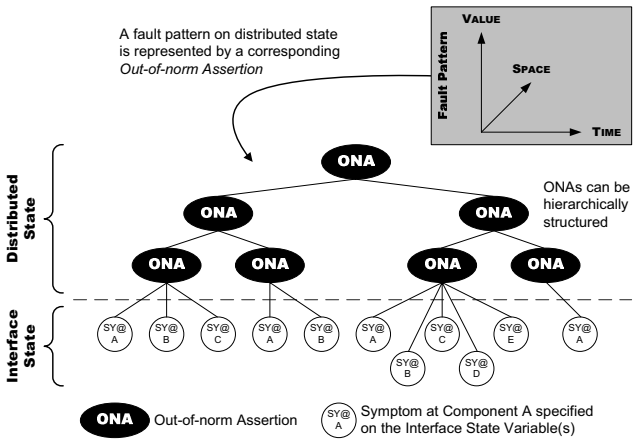

Figure 3. Definition of Out-of-Norm Assertion

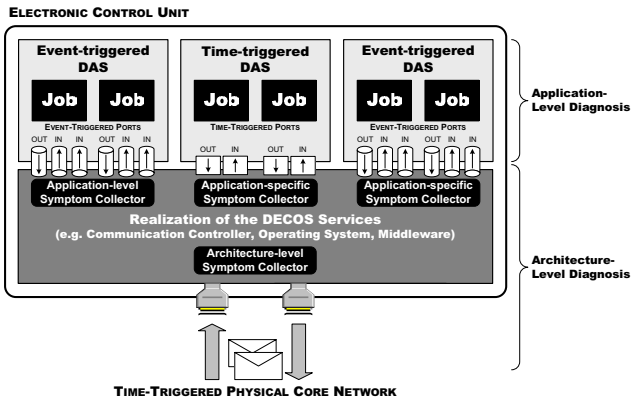

Figure 4. Architecture-Level and Application-Level Symptom Collection

the interface state of a different component. As depicted in Figure 3 ONAs can be hierarchically structured. This allows for the exploitation of identified symptoms for the implementation of different ONAs. ONAs operate solely on the *interface state*. This way, the internals of jobs remain hidden. This ensures two main requirements imposed by industry, the protection of intellectual property and no need to change existing application code.

ONAs do not provide a definite classification whether a component is correct or incorrect in case only a subset of the specified symptoms fires. In this case, we speak of an *anomaly*, i.e. we cannot ascribe the behavior of the component to a specific fault pattern. In order to decide on the maintenance action for a component, an assessment over time is necessary. The repeated evaluation of evidence gathered by ONAs provides the foundation for the analysis process that ultimately decides whether a component is correct. ONAs are gathering evidence in order to decide on a particular pattern affecting the state of the system. This process can be compared with a gathering evidence of different diagnostic techniques in medicine (e.g., temperature measurement, computer tomography, x-ray). In case sufficient evidence is gathered, a suspicion for a particular disease is confirmed or falsified.

3 The Diagnostic Framework

In order to integrate diagnosis into the development process, a framework that supports both, the application developers and system designers is needed. Such a framework also helps the developers to precisely specify the diagnostic checks and to treat diagnosis not as an addendum but as an integral part of all development phases. By applying a framework, additional design faults can be avoided by providing the possibility to automatically transform the symptom detectors and analysis specifications (i.e. the ONAs) into executable code that can be executed as part of the high-level services.

3.1 Symptom Collection on the Local Interface State

As defined in Section 2.3.3, symptoms are the local manifestation of fault patterns at a single component. Since the diagnostic framework supports architecture-level and application-level diagnosis, symptoms for both types need to be supported. Typically, application-level symptoms are defined on the ports of jobs and monitor all incoming and outgoing messages, architecture-level symptoms are deployed for the detection of failures at the core and high-level services of the integrated architecture. By correlating the information from both, application-level and architecture-level symptoms a finer differentiation between hardware and software faults is possible, thus overcoming today's ECU centered diagnosis schemes.

Architecture-level Symptom Collectors. As indicated in Figure 4 the architecture-level symptom collector is independent of any job-specific application logic. These checks include for example the state of the membership, the clock correction term, or the health state of the connection to the physical network (e.g., message reception on all replicated physical channels). In addition, architecture-level symptom collectors are applied to gather diagnostic information originating from the operating system (e.g., partitioning failures) or high-level services (e.g., queue overflows at the virtual network service). This enables reuse of the specified checks in a large variety of systems in case the symptom has proven in the field to provide valuable diagnostic information. Architecture-level diagnostic checks do not need to include any knowledge about the physical or functional structure of the system. This is solely encoded into the ONA processing the information provided by the symptom collectors.

Application-level Symptom Collectors. While architecture-level symptoms are checks on interface

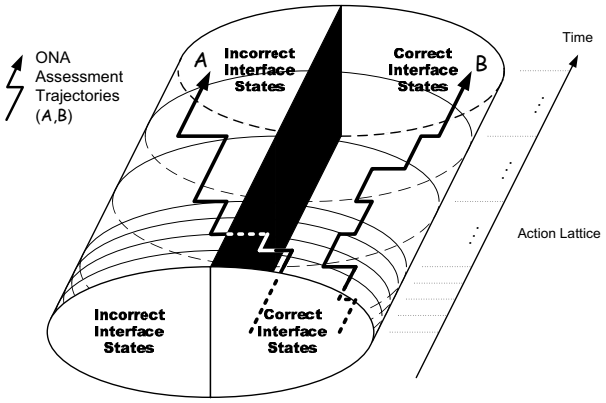

Figure 5. Assessment Process

state variables of a component, application-level symptoms are evaluated against the port state of the jobs hosted on a component. Thus, an application-level symptom collector monitors the behavior of the job at one or more ports of a link of the respective virtual network [9]. In contrast to architecture-level symptoms, application-level symptoms highly depend on the application context and need thus be devised by the application developers. Only the deep understanding on the dynamics of the application and relations between the different jobs allows defining meaningful symptoms.

3.2 Transport of Diagnostic Information

Whenever a symptom collector detects a violation of a linking interface specification, the symptom collector sends a diagnostic message to the analysis subsystem via a so-called *virtual diagnostic network*. This 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* is introduced [5].

3.3 Analysis

The analysis of the gathered diagnostic information (i.e. symptoms) is realized in a designated DAS – the diagnostic DAS – to ensure that the diagnostic subsystem cannot interfere with jobs of other DASs and to not restrict the choice of implementations (e.g., central diagnostic component vs. distributed solution). As defined in Section 2.3.3 and illustrated in Figure 3 an ONA combines information from the value, time, and space domain in order to improve the accuracy of the analysis process according to previously introduced maintenance-oriented fault model.

Value Domain. Checks regarding the value domain are handled by the symptom collectors on the interface state of a component or job as described in Section 3.1. Typically, knowledge about the physical interrelationships is used to define meaningful value domain checks (e.g., threshold values, allowed range of values) and allows for example to relate values from different sensors (e.g., plausibility checks for a state estimation model).

Time Domain. The evaluation process performed by the diagnostic DAS is illustrated in Figure 5. The evaluation process is based on a consistent notion of state, which is provided through the *action lattice* of the sparse time base established by the core services [14]. The arrows in Figure 5 indicate the assessment trajectories. At first both arrows show component conformance with the specification, i.e. correct interface states. As time progresses arrow A exhibits an increasing confidence for a violation of the specification, while arrow B indicates a component behavior in accordance with the specified service. In particular, by exploiting the time domain, a distinction between transient (i.e. non-destructive external) and intermittent (i.e. internal) hardware faults is possible [15].

Space Domain. The introduced integrated architecture provides a finer granularity of diagnostic information than federated systems. The assessment process exploits this knowledge about the functional and physical structure of the integrated architecture. The decomposition of the overall system into DASs with respective jobs is a key element for a more precise differentiation of experienced faults. By including the three dimensions of time, value, and space into the judgment process, a discrimination into internal hardware faults, external hardware faults and software faults is possible [4]. For instance, consider the system depicted in Figure 6. In case a software fault hits the jobs A_1, A_2, and A_3 of the non safety-critical DAS A, the fault effects only the DAS A, since the error containment mechanism of the architecture ensures that this fault cannot propagate to other DASs. In contrast, in case a hardware fault hits a component hosting multiple jobs of different DASs, it is very likely that the impact of this fault is not limited by DAS borders. A hardware fault will cause multiple jobs hosted on one component to fail (e.g., the jobs A_3, C_1, C_2, and S_2 on component 2 in Figure 6).

The recognition of correlated job failures is also important in the detection of faults affecting architecture supported fault-tolerance mechanisms, such as Triple Modular Redundancy (TMR) mechanisms. This fault-tolerance mechanism is characterized by the replication of identical jobs on three different components in order to tolerate single hardware faults. In case the jobs S_1, S_2, and S_3 are forming a TMR system, the spatial dimension of an ONA covering deviations in the services of the three replicas spreads across components 1, 2, and 3 (since a component is the FRU with respect to hardware faults). In case one of the replicated safety-critical jobs fails, an analysis if correlated failures of

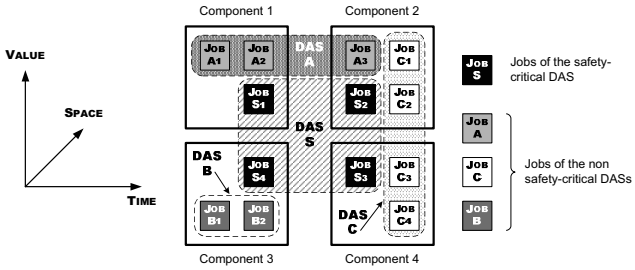

Figure 6. Judgment According to the Three Dimensions: Time, Value and Space

jobs of other DAS executed at the same time on the same component exist will supply evidence whether a hardware fault effects the component.

4 Specification and Execution of Out-of-Norm Assertions and Symptom Collectors

Since ONAs encode fault patterns on the distributed state of the system, methods that allow expressing characteristics in the value, time, and space domain. In the proposed framework we use timed automata for both the specification of symptoms (i.e. component/job local checks on the interface state) and the consecutive analysis process. The underlying sparse time base allows the precise definition of execution semantics that are also subject of discussion in this section.

A timed automaton [16], i.e. a state transition graph annotated with timing constraints, has an intuitive syntax and semantic that makes it especially interesting in the specification and design of diagnostic algorithms. Furthermore, using timed-automata as the specification method has also the significant advantage of having a representation that can easily be transformed into a machine executable form.

4.1 The Timed Symptom/Analysis Automaton

Since the timed automata used for the specification of ONAs need to be executable, we restrict timed automata defined by [16] to be nonzeno (i.e. there is no infinite sequence of transitions without any progression of time in between), deterministic and require that the invariant of each location is complementary to the guards of all outgoing edges, i.e. an edge must be taken as soon as possible. In the following we extend the definition by [16] to be suitable for our purposes.

Guards and Actions. In order to formally define the timed symptom/analysis automaton, we specify what constraints are allowed as the enabling conditions called *guards* of a timed automaton. We define the set $\Phi(X, V, M) : (X, V, M) \mapsto (\mathbb{T}, \mathbb{F})$ of guards for a set of clocks X, variables V, and messages M via the following grammar:

$$\varphi := \quad x \circ c \mid x \circ v \mid x \circ m \mid v \circ c \mid v \circ m \mid \\ v \circ v' \mid m \circ m' \mid \text{av}(m) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in X, c is a constant in $\mathbb{N}$, v and v' are variables in V, m and m' are messages in M. $\circ$ is a binary relation $(\leq, <, =)$. $\text{av}(m)$ tests whether a message m is available at the respective input port (only needed for event-triggered communication).

The set of *actions* $\Lambda(X, V, M) : (X, V, M) \mapsto (X, V, M)$ is defined via the following grammar:

$$\lambda := \quad x := c \mid x := m \mid v := c \mid v := m \mid v := v' \mid \\ m := c \mid m := v \mid m := m' \mid m_{\text{diag}}! \mid m? \mid \lambda_1; \lambda_2,$$

where x is a clock in X, c is a constant in $\mathbb{N}$, v is a variable in V, and m is a message in M. $m_{\text{diag}}! \in M$ is a diagnostic message to be disseminated via the virtual diagnostic network, and $m?$ reads message m from the respective input port. Note, that in case of event-triggered communication $m?$ is consuming, in contrast to $\text{av}(m)$.

Definition of the Syntax. A timed symptom/analysis automaton A is a tuple $\langle L, L^0, X, V, M, I, E\rangle$, where

- L is a finite set of locations,
- $L^0 \in L$ is the initial location,
- X is a finite set of clocks,
- V is a finite set of variables,
- M is a finite set of messages,
- I is a mapping that assigns to each location an invariant as a constraint in $\Phi(X, V, M)$ $(I : L \mapsto \Phi(X, V, M))$,
- $E \subseteq L \times \Phi(X, V, M) \times \Lambda(X, V, M) \times L$ is a set of transitions. A transition $\langle s_1, \varphi, \lambda, s_2\rangle$ represents an edge from location s_1 to location s_2 with guard φ. The guard is a constraint of clocks X, variables V and messages M and determines when the transition is enabled. The action λ is a set of assignment and message operations to be performed.

4.2 Execution of the Timed Automata

The timed symptom detection/analysis automata are executed on the action lattice of the sparse time base. In the silence interval with respect to the communication services both the timed automata for symptom detection and analysis as part of an ONA are executed. The algorithm is presented in Figure 7 and works as follows:

1. The time variable is set to the beginning of the last interval of activity of the sparse time base with respect to the action lattice for virtual network service $T =$

```
clocks              X
variables           V
messages            M
current location    s

T = T_activation − n
while ( T < T_activation )
   while (∃ ⟨s_1, φ, λ, s_2⟩ ∈ E with φ = 𝕋 ∧ s = s_1)
      (X, V, M) := Λ(X, V, M)     //execute action
      s := s_2                    //new location
   end
   T = T + 1                      //advance time
   ∀x ∈ X : x = x + 1
end
```

Figure 7. Execution Step (n ticks) of the Timed Automaton A

$T_{activation} - n$, where $T_{activation}$, is the actual global time. n denotes the number of ticks elapsed during on interval of silence and activity.

2. As long as the simulation time T is smaller than the actual time $T_{activation}$, the execution continues. In case this condition does not hold, the execution of this automaton is terminated for this activation cycle.

3. A transition is taken whenever a guard φ is enabled. In case no guard evaluates to $\mathbb{T}$, the following step (4) is skipped.

4. If a transition is taken, all corresponding actions λ are executed (e.g., updating of variables, message reception) and the current location is updated.

5. Both the global simulation time T and local clock variables X are increased by 1. Continue with step 2.

5 Realization of the Diagnostic Services for Borderline Faults

In the following section we exemplarily describe how the introduced ONAs are implemented in the context of the integrated DECOS architecture. We thereby focus on the detection and analysis of component borderline faults according to the fault model introduced in Section 2.3.1.

Wiring plays a central role in any distributed system environment, since it provides the infrastructure for exchanging the data between components. Like any other part of the system, the electrical interconnection system is exposed to environmental stress as well as assembly and design faults. Considering that a typical middle-class car has about 40 ECUs and approximately 800 wires [7], the likelihood of connector problems is high. In fact, recent studies [17] indicate that 30% of electrical failures can be attributed to connector problems. Wiring problems, especially connector problems, are difficult to detect, since the inspection itself can be the corrective action (e.g., loose contacts). For this reason, it is important to provide means for the online detection of connector failures.

5.1 Implementation Platform

Our prototype implementation of the DECOS architecture consists of a cluster of five components using TTP/C [18] as the time-triggered core communication service. On each system component the embedded real-time Linux variant Real-Time Application Interface (RTAI) [19] is employed for establishing an encapsulated execution environment for hosting multiple jobs on one physical component. For this partitioning the capabilities of the RTAI/LXRT extension are heavily utilized. For more information on the used hardware and software platform see also [12].

5.2 Detection of Component Borderline Symptoms

The TTP/C controller determines the frame status for each received frame [18]. Depending on this message status field, the application can read or discard the received message. This so-called *Error Indication Field* allows an analysis of the status of the communication channels. The TTP/C controller compares the frames from both channels and declares the received frame as correct as long as one frame is correct. This strategy is suitable for application transparent fault-tolerance, however, an integrated diagnostic solution must take the information from both channels into account to enable an investigation of possible physical faults of the replicated channels or bus drivers. In TTP/C we can use the frame status information as described in the following for judging whether a borderline fault is affecting the system. As shown in Figure 9, TTP/C differentiates between Correct Frames, Invalid Frames (i.e. syntactically incorrect frames), Null Frames (i.e. no activity on the channel, not even noise), and Incorrect Frames (i.e. wrong Cyclic Redundancy Code (CRC)) [18].

In Figure 8 a timed automaton is depicted that implements the symptom detection on the frame status of each physical channel. The timed automaton performs the check on the status of the network in each TDMA slot (using clock variable x) according to the cluster schedule of the core network. Whenever, a new TTP frame is received, the execution of the timed automaton progresses and the frame status is evaluated. In case the frame status of channel 0 or 1 is other than *correct*, a corresponding diagnostic message is constructed and forwarded to the virtual diagnostic network. This send operation is specified using $m_{(\text{type,symptom\#},u.c.)}!$ in the automaton.

By replacing the generic check φ_3 stated in Figure 8 commands for accessing the respective registers of the physical communication controller the timed automaton can be executed on the target platform.

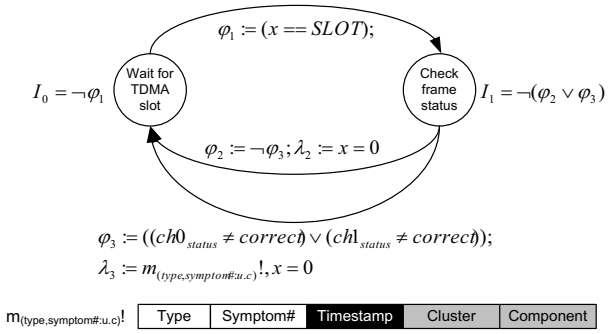

Figure 8. Symptom for Component Borderline Faults

5.3 Transport of the Diagnostic Message

Once a symptom has been identified, this information is encoded into a diagnostic message and forwarded to the analysis DAS, responsible for executing the ONA that combines the information to judge on the type of fault affecting the system. The message content for the transport of borderline symptoms is described in Figure 9. The message includes information on all three domains as elaborated on in Section 2.3.2.

Field Name	Semantics
Message Type	This field is set to `TYPE_ARCHITECTURE` in order to differentiate the message from application-specific symptoms.
Symptom ID	The unique symptom ID is set to `SYM_CORRECT_CH[0/1]`, `SYM_NULLFRAME_CH[0/1]`, `SYM_INVALID_CH[0/1]`, `SYM_INCORRECT_CH[0/1]`, `SYM_OTHER_CH[0/1]`, `SYM_TENTATIVE_CH[0/1]` or `SYM_UNKNOWN_CH[0/1]`.
Time Domain	This field includes timestamp of the global time base.
Space Domain	This field contains the identification of the physical cluster and component.

Figure 9. Diagnostic Message

5.4 Analysis: Determination of Component Borderline Faults

The analysis of the gathered diagnostic information (i.e. symptoms) is shifted into a designated DAS – the diagnostic DAS – to ensure that the diagnostic subsystem cannot interfere with jobs of other DASs. In our implementation we extend a simple threshold-based algorithm for the analysis such as the α-count algorithm [15]. The rationale for the α-count mechanism is to decide on the point in time when keeping a system component on-line is no longer beneficial. The algorithm is partly based on the observation that intermittent (transient internal) faults exhibit a relatively high occurrence rate after their first appearance [4]. The α-count is a threshold-based fault classification mechanism designed to identify permanent faulty components from components affected by external transient faults. The

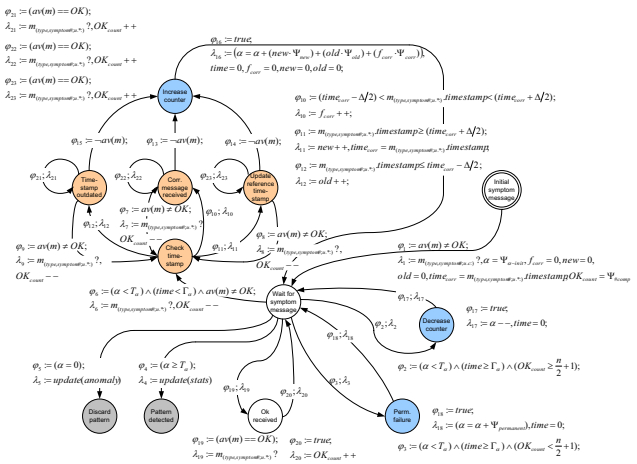

Figure 10. Determination of Component Borderline Faults

main idea of the algorithm is to keep track of every fault occurrence in each component. When the α-counter value exceeds a given threshold value, the component is diagnosed as affected by a permanent/intermittent fault. Depending on the expected frequency of permanent, intermittent and transient faults the values assigned to the parameters of the algorithm are set.

However, in contrast to typical analysis algorithms processing only local information, the timed analysis automaton as shown in Figure 10 for the systemic diagnosis of borderline failures correlates information from other components of the cluster to indisputably judge whether a failure has occurred. In case no correlated symptoms have been detected the increase of the α-counter value is marginal. By contrast, once a correlated information is detected, the value is multiplied with the weighting factor Ψ_{corr} to emphasize the importance of such correlated information. This way the α-counter is increased by a significant higher value compared to the case when only a single symptom has been detected.

Since in our prototype implementation a core communication network with two replicated channels is used, for each component two ONAs for the determination of borderline faults are required (i.e. one for each channel).

5.4.1 Definition of Thresholds and Weighting Factors

Every threshold-based analysis technique depends on the setting of the parameters that determine the behavior of the algorithm [15]. We use the following parameters in our prototype implementation

- the penalty weighting factors ($\Psi_{\text{new}}, \Psi_{\text{old}}, \Psi_{\text{corr}}, \Psi_{\text{permanent}}, \Psi_{\alpha-\text{init}}$)
- the α-counter threshold (T_α), and
- the timing parameters (Γ_α, Δ).

As shown in Figure 10 the threshold value T_α determines when a fault pattern is detected and stored in the analysis data structures (the action $\lambda_4 :=$ update(stats)). The parameter Γ_α defines the length of the time interval between decreasing the α-counter value in case no symptom message is received. The constant Δ is used for specifying the time window during which correlated information with respect to the previously received symptoms is processed. Whenever a symptom with a timestamp is received that strengthens the belief in the correctness of the previously received symptom(s) the variable f_{corr} is increased, the holds the number of correlated messages. The value of f_{corr} is multiplied by the weighting factor Ψ_{corr} before adding to the α-counter value to amplify the impact of diagnostic information that has been confirmed by other components of the system (i.e. following the cross-checking principle [13]). After processing all incoming symptoms the α-counter value is updated according to the transition enabled by guard φ_{16} and action λ_{16}:

$$\alpha = \alpha + (new \cdot \Psi_{\text{new}}) + (old \cdot \Psi_{\text{old}}) + (f_{\text{corr}} \cdot \Psi_{\text{corr}})$$

where, new holds the number of newly received messages (with future timestamp), old holds the number of received messages with outdated timestamp (due to possible delays introduced by the event-triggered virtual network service). The weighting factors Ψ_{new} and Ψ_{old} are used to define the impact of outdated symptoms (stored in the variable old) and new symptoms (stored in the variable new).

5.4.2 Implementation of the Timed State Machine

In the following we discuss relevant implementation details of the state machine described in Figure 10.

Global Time. Since the progression of the execution trace of the automaton implementing the analysis algorithm depends on the progression of real-time, all local clock variables are synchronized with the global time provided by the underlying time-triggered core network. As already stated, the 16 bit global time provided by TTP is extended to a node local 32 bit time field to implement timing constraints exceeding 16 bit (or approximately 330 ms in the used schedule and TTP cluster configuration). Of particular interest with respect to the progression of real-time is state `CHECK_TIME_STAMP`, where the following timing analysis is performed:

- In case $\varphi_{10} := (\text{time}_{\text{corr}} - \frac{\Delta}{2}) < m_{(\text{type,symptom\#},u.*:)}.\text{timestamp} < (\text{time}_{\text{corr}} + \frac{\Delta}{2})$ is enabled, a correlated diagnostic message has been detected and the correlation factor f_{corr} is increased.
- In case $\varphi_{11} := m_{(\text{type,symptom\#},u.*:)}.\text{timestamp} \geq (\text{time}_{\text{corr}} + \frac{\Delta}{2})$ evaluates to $\mathbb{T}$, a message with a new symptom has arrived. As a consequence the new counter is increased and the variable holding the reference timestamp $\text{time}_{\text{corr}}$ is updated accordingly.
- In case $\varphi_{12} := m_{(\text{type,symptom\#},u.*:)}.\text{timestamp} \leq (\text{time}_{\text{corr}} - \frac{\Delta}{2})$ a message with an outdated timestamp has arrived. This condition might fire, due to possible message delays at the virtual event-triggered network. If the guard φ_{12} evaluates to $\mathbb{T}$, the old counter is increased and later added to the α-counter value.

Although a node local 32 bit timestamp is used in the implementation, a time overflow might occur. This has been taken into account in the prototype implementation.

Event Semantics. The use of event semantics for encoding diagnostic information has the advantage of effective usage of the available bandwidth for diagnosis. Since only changes in interface state variables are distributed, a failure is active as long as no message is received, stating that a correct state of the variable has been restored. Whenever no confirmation messages arrive, a permanent failure is assumed. This issue has to be taken into account when implementing the analysis algorithm.

Therefore, the variable OK_{count} holds the number of received messages indicating that the status of the channel is correct. If the message data field contains the symptom `SYM_CORRECT_CH0/1` the variable OK_{count} is incremented, otherwise decremented (see also the transitions φ_6, λ_6 and $\varphi_{19}, \lambda_{19}$ in Figure 10). In the correct case OK_{count} equals the number of components in the system. Consequently, the function `av(m)` needs to return the symptom value for implementing this strategy. As depicted in the timed automaton, in case the value of OK_{count} is smaller than the number of the deployed physical components, there are two possible transitions:

- If $OK_{\text{count}} \geq \lfloor \frac{n}{2} \rfloor + 1$ evaluates to $\mathbb{T}$, then the majority of the nodes in the cluster have confirmed that the channel status is correct again. In case OK_{count} does not equal the number of components, the counter is increased over time. In the implementation every $5 \cdot \Gamma_\alpha$ time units, OK_{count} is incremented by one to compensate for missing "OK messages".
- If $OK_{\text{count}} < \lfloor \frac{n}{2} \rfloor + 1$ evaluates to $\mathbb{T}$, then only a minority of the nodes in the cluster have confirmed that the physical connection to the time-triggered core network has been reestablished. In this case a permanent failure has been detected and the α-counter value is increased by $\Psi_{\text{permanent}}$ every Γ_α time units as long as either the missing "OK messages" arrive or the threshold T_α is exceeded.

Exceeding the Threshold T_α. In case the α-counter value exceeds the threshold T_α, the global analysis data structure is updated accordingly. Then, the timed automaton is reset and the execution starts over again. Reoccurring overstepping of the threshold then results in additional ONA firing and thus strengthens the belief in the previous analysis results. However, alternative strategies can be implemented.

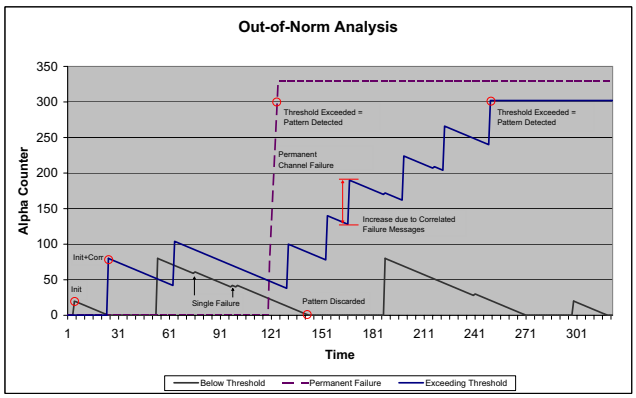

Figure 11. Alpha Counter Strategies

For example, after exceeding the threshold T_α the component can be declared as permanent faulty and scheduled for maintenance action.

Decrease of the α-counter. In case $\alpha \leq 0$ the fault pattern for a channel fault is discarded. Since the diagnostic architecture follows the record any single anomaly design principle, the anomaly counters for the respective component are increased. Subsequently, the timed automaton is reset and the execution starts over again by initializing the automaton data structures. The idea behind introducing an anomaly counter is the strategy that a high anomaly counter can be used as engineering feedback for a possible pending problem.

5.4.3 Parameter Settings

In the implementation of the threshold-based analysis algorithm depicted in Figure 10 the following values for the parameters are used: Γ_α is set to an interval of 30 seconds. In case no further symptom is received within 30 seconds, the α-counter value is decreased by one. In the prototype configuration, messages with a timestamp differing no more than one TDMA round are assumed to be correlated. This opens a time window for processing correlated information. A single symptom increases the α-counter value only by 2 as defined Ψ_{new}, whereas any additional correlated symptom message increases the α-counter value by 30 corresponding to the parameter Ψ_{corr}. The parameter $\Psi_{\alpha-\text{init}}$ defines the penalty for the first occurrence of a symptom and is set to 20. $\Psi_{\text{permanent}}$ increases the α-counter value by 50 every Γ_α time units in case a permanent channel failure is detected.

5.4.4 Implementation Results

Figure 11 depicts the measurement results of experiments analyzing the performance of the implemented α-counter strategy. On the basis of three chosen representative fault types, namely

- Permanent internal borderline failure (denoted as "Permanent Failure")
- Transient internal borderline failure (denoted as "Exceeding Threshold")
- Transient external borderline failure (denoted as "Below Threshold")

we describe the measurement curves of the α-counter values in more detail. The x-axis of the diagram presented in Figure 11 shows the progression of time. Each time unit corresponds to Γ_α, i.e. the time that has to elapse without a the reception of a symptom message to decrease the value of the α-counter. The y-axis represents the value of the α-counter.

Permanent internal channel failure. As depicted, once a permanent channel failure occurs, the value of the α-counter is increased by $\Psi_{\text{permanent}}$ every Γ_α macroticks. Since no message with symptom `SYM_CORRECT_CH0/1` is received, indicating that the physical link is successfully established again, and the majority of the components in the cluster support this view ($OK_{\text{count}} < \frac{n}{2} + 1$), an increase of the α-counter value exceeding T_α is the result. Consequently, the channel is declared as being permanent faulty and a corresponding entry into the diagnostic data structures is written.

Transient internal channel failure. The diagnosis of communication blackouts on the deployed bus systems is of critical importance. Since transient internal channel failures require maintenance action such as inspection of the cable loom or change of a defective ECU, accurate diagnosis is vital to keep warranty costs low. The following points as highlighted in Figure 11 are of special interest. At $t = 23$ a channel failure is detected and processed, resulting in an increase of $\Psi_{\alpha-\text{init}}$ of the α-counter value. Since correlated failure messages confirm this channel failure, the term $new \cdot \Psi_{\text{new}} + f_{\text{corr}} \cdot \Psi_{\text{corr}}$ is added to the value of the α-counter. As depicted, the continuous detection of symptoms indicating a potential internal channel failure causes that the fault pattern is never discarded. In fact, the reoccurring correlated symptom detections result in an increase of the α-counter value beyond T_α as time progresses.

Transient external channel failure. According to the maintenance-oriented fault model, transient external faults are falling into the category of faults where no maintenance action is required to restore the intended functionality. Whenever, the first diagnostic message with a symptom that is relevant for the analysis of component borderline faults is processed the α-counter is increased according to $\Psi_{\alpha-\text{init}}$. The curve presented in Figure 11 shows this increase in the beginning, due to an unconfirmed symptom message. Whenever the α-counter value, that decreases over time, is 0 again, the fault pattern is discarded and the

timed automaton reset after writing an entry into the analysis data structures. Then the execution of the timed automaton restarts in state "initial symptom message". Note, that although the ONA never fires, i.e. the threshold T_α is never exceeded by the α-counter value, every entry made into the analysis data structures whenever the fault pattern is discarded is an indication for an anomaly. This data can provide important feedback when analyzing a large population of maintenance records.

6 Conclusion

Integrated architectures offer the possibility for diagnosis exceeding state-of-the-art solutions by operating on the distributed state in order to exploit knowledge about the physical and functional structure of the system in the analysis process. In this paper we introduced a framework that supports execution of so-called Out-of-Norm Assertions (ONAs) in order to reduce the fault-not-found ration in electronic systems. Timed automata are used to specify both the local detection mechanisms as well as the global analysis algorithms that combine the gathered diagnostic information in the space, time, and value domain in order to decide on a particular maintenance action. A prototype implementation has shown the feasibility and effectiveness of this strategy.

Acknowledgments

This work has been supported in part by the Austrian Advanced Automotive Technology Project under project No. 809437 and the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

References

[1] J. Barkai, "Vehicle diagnostics–are you ready for the challenge?" in *Proc. of Automotive & Transportation Technology Congress*, 2001.

[2] D. Thomas, K. Ayers, and M. Pecht, "The 'trouble not identified' phenomenon in automotive electronics," *Microelectronics Reliability*, 2002.

[3] M. Pecht and V. Ramappan, "Are components still the major problem: a review of electronic system and device field failure returns," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, 1992.

[4] C. Constantinescu, "Impact of deep submicron technology on dependability of VLSI circuits," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2002.

[5] J. Gait, "A probe effect in concurrent programs," *Software Practice and Experience*, 1986.

[6] H. Kopetz and N. Suri, "Compositional design of RT systems: A conceptual basis for specification of linking interfaces," in *Proc. of 6th ISORC*, 2003.

[7] P. Peti, R. Obermaisser, F. Tagliabo, A. Marino, and S. Cerchio, "An integrated architecture for future car generations," in *Proc. of the 8th ISORC*, 2005.

[8] P. Peti, "Diagnosis and maintenance in an integrated time-triggered architecture," Ph.D. dissertation, Vienna University of Technology, 2005.

[9] R. Obermaisser, P. Peti, and H. Kopetz, "Virtual networks in an integrated time-triggered architecture," in *Proc. of 10th IEEE WORDS*, 2005.

[10] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," LAAS-CNRS, Tech. Rep., 2001.

[11] P. Peti, R. Obermaisser, and H. Kopetz, "A maintenance-oriented fault model for the DECOS integrated diagnostic architecture," in *Proc. of the WP-DRTS*, 2005.

[12] B. Huber, P. Peti, R. Obermaisser, and C. E. Salloum, "Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture," in *Proc. of the 3rd WISES Workshop*, 2005.

[13] P. Peti, R. Obermaisser, and H. Kopetz, "Out-of-norm assertions," in *Proceedings of the 11th RTAS*, 2005.

[14] H. Kopetz, "Sparse time versus dense time in distributed real-time systems," in *Proc. of 12th ICDCS*, 1992.

[15] A. Bondavalli, S. Chiaradonna, F. D. Giandomenico, and F. Grandoni, "Threshold-based mechanisms to discriminate transient from intermittent faults," *IEEE Transactions on Computers*, 2000.

[16] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, 1994.

[17] J. Swingler, J. McBride, and C. Maul, "Degradation of road tested automotive connectors," *IEEE Transactions on Components and Packaging Technologies*, 2000.

[18] H. Kopetz, *Specification of the TTP/C Protocol*. TT-Tech, 1999.

[19] D. Beal et al., "RTAI: Real-Time Application Interface," *Linux Journal*, Apr. 2000.