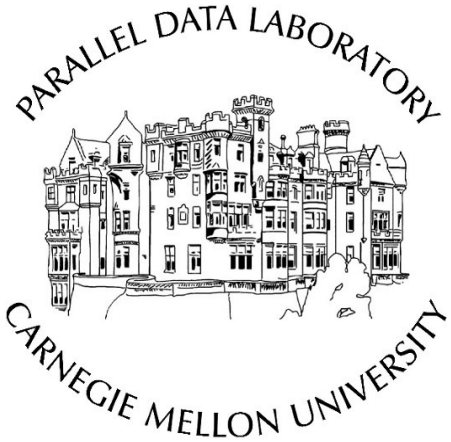

Diagnosis in Automotive Systems: A Survey

PATRICK E. LANIGAN, SOILA KAVULYA and PRIYA NARASIMHAN
Carnegie Mellon University
THOMAS E. FUHRMAN and MUTASIM A. SALMAN
General Motors Research & Development

CMU-PDL-11-110
June 2011

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Modern automotive electronic control systems are distributed, networked embedded systems. Diagnostic routines implemented on individual components cannot adequately identify the true cause of anomalous behavior because their view is restricted to component-local information. A growing trend in diagnostics research for these systems is to use system-level approaches to diagnose anomalous behavior and provide a consistent, global view of the system's health. Current approaches are typically motivated by a desire to improve either off-line maintenance or run-time safety.

Acknowledgements: This research was supported in part by General Motors Company.

Keywords: diagnosis, automotive, safety, agreement, consensus, maintenance

1 Introduction

Automotive systems are safety-critical and are required to be highly fault-tolerant in operation. Automotive systems consist of mechanical, hydraulic, software and hardware components. There is a staggering amount of embedded computing within automotive systems. For instance, current GM vehicles contain dozens of microprocessors and dozens of megabytes of software.

Software-intensive distributed electronic control systems are increasingly being used in the automobile industry to provide convenience and safety features to vehicle drivers and passengers, with increasing levels of automation and control authority. A growing trend is to assist the driver in maintaining safe control over the motion of the vehicle under a variety of conditions that include congested traffic conditions, adverse weather and road conditions, varying states of health of vehicle equipment, and varying skill levels of drivers. Previously such driver assistance has been provided in the form of information or warnings to the driver, but increasingly such assistance will be provided by actively manipulating actuators that control vehicle longitudinal acceleration and deceleration, lateral position, and vertical displacement. The long term trend is towards partial or even fully autonomous operation of a single vehicle, or even of fleets of vehicles. One example of autonomous vehicle operation is the BOSS vehicle developed for the recent DARPA Urban Challenge [57].

These advanced convenience and safety features must be designed to tolerate faults to ensure the safety of the driver, passengers, and surrounding people and objects such as pedestrians and other vehicles. Typically a systematic safety analysis is conducted offline during the design phase, with the vehicle not being operational, to evaluate both the severity and likelihood of the consequences of possible faults. Faults leading to potentially undesirable consequences must be detected, and the appropriate mitigating action must be taken to ensure safety. Frequently the mitigating actions are categorized as either being either fail-safe (meaning that the system is shut down into a safe state) or fail-operational (meaning that the system must maintain at least some level of continued functionality to ensure safety, possibly with degraded performance). In either case, before the mitigating action can be taken, the system must be capable of detecting that the fault is present in the first place. Furthermore, this detection of faults must take place within a specified amount of time (referred to as the detection latency), to allow enough time for the mitigating action to maintain safe dynamic behavior of the vehicle motion, and with specified coverage (referred to as the detection coverage), for a fault that is not detected can generally not be mitigated!

Failure diagnosis goes beyond fault detection by providing extended information on the underlying cause of a system failure. Failure diagnosis is distinguished from fault detection in that detection aims mainly to determine that *some fault occurred*, while failure diagnosis might reveal *what kind of fault occurred*, *what component(s) is/are responsible*, and *what caused the fault*. Depending on the goals of the specific failure diagnosis strategy, this extended information could be useful to engineers during design, integration and testing; technicians during service; and at runtime to allow for more robust failure mitigation mechanisms.

Most of the work in the area of failure diagnosis for automotive systems has focused on offline diagnostics performed by service technicians or mechanics in the field, often to discover which parts ought to be replaced and/or sent back for repair or warranty claims. Even Diagnostic Trouble Codes (DTCs), though detected online during vehicle operation, are intended for use by a service technician after being read out using a scan tool for the purpose of offline maintenance. *Warranty Week* reports that warranty costs for automotive manufacturers can be prohibitively high, and can arise from multiple issues: faulty parts, invalid claims, customer dissatisfaction, part replacements, etc. For just the three passenger-car companies, General Motors, Ford and DaimlerChrysler, the total warranty claims topped $15.34 billion, on sales of some 20.8 million units in 2006 [61]. Given these fairly high costs, it seems reasonable that most of the effort on failure diagnosis has been spent thus far on offline troubleshooting techniques to reduce the number/cost of warranty claims and part replacements.

However, autonomous driving vehicles challenge this perspective. These vehicles must continue to operate, keeping the occupants of the vehicle safe under all circumstances, while being able to adapt to adverse and unforeseen condition. A study of BOSS, the DARPA Urban Grand Challenge winner, reveals that some failure modes of operation were seen at runtime that could not be tested for or anticipated at design time [8], simply because of emergent road conditions, environmental factors, etc. Autonomous driving systems can simply not afford offline diagnostics; they must support online diagnostic capabilities to quickly get to the root-cause of any anomalous situation, to understand whether the anomaly represents a failure or an emergent (but safe) condition, and to adapt the system accordingly, all the while preserving safety, fault-tolerance and continuous operation. There is another challenging aspect to online diagnosis for automotive systems — today's offline diagnosis is performed by expert service technicians who have hours of experience to enable them to diagnose the root-cause in the field, albeit on a non-operational vehicle. Autonomous driving systems must be able to perform such diagnostic procedures, completely unassisted by technicians, without access to years' worth of field data or prior diagnostic experience. Building such self-diagnosing, self-repairing autonomous driving systems is a challenge indeed.

1.1 Outline and Scope

This paper surveys the technical literature for failure diagnosis approaches suitable for the emerging class of distributed, embedded control-systems for convenience, safety, and autonomous operation of automobiles. This survey will include the relevant fault models, failure effects or manifestations, fault injection techniques used in developing and validating the safety system, requirements for failure diagnosis, and finally the actual failure diagnosis methods themselves.

This survey addresses both the online failure diagnosis approaches necessary to maintain the safe operation of the vehicle at operation time, as well as the offline failure diagnosis approaches suitable for maintenance of the vehicle in a service facility. Also, this survey focuses on faults in the embedded "control system" itself (the computers, communication links, memories, software, sensors, and actuators that implement the convenience, safety, and autonomous driving features), rather than faults in the mechanical objects being controlled (often referred to as the "plant"). More importantly, this survey provides a candid and critical look at the current state-of-the-art, highlighting where the literature or field practices fall short of accomplishing the goals of safety-critical autonomous driving systems, and where the research challenges remain.

2 Background

In this section, we will explain some of the terms that are central to our discussion of failure diagnosis. Our purpose here is not to develop a new or comprehensive taxonomy, but to provide clarity for the remainder of this survey.

2.1 Automotive architectures

Automotive systems consist of mechanical, hydraulic, software and hardware components that implement specific vehicle functions. This paper focuses exclusively on the embedded computing architectures that provide the foundation for vehicle control systems. Such architectures encompass distributed control and information systems implemented on multiple Electronic Control Units (ECUs) interconnected by serial communication busses, together with sensors and actuators locally connected to the I/O ports of the ECUs or to sensor/actuator busses. These architectures are typically highly distributed, for two primary reasons: (1) the failure of one component allows functions implemented on other components to continue operating, and (2)

proximity to the sensors providing input information, and actuators being controlled, minimizes wire length and wire-harness complexity.

Frequently automotive control systems are categorized as either event-based or time-based. Infrequently occurring or sporadic functions, such as locking or unlocking of doors or turning headlamps or turn-signals on or off, are event-based and accordingly communicate information over serial busses only when variables change state, while continuously operating functions such as steering control or fuel injector control are time-based and accordingly communicate continuously varying information at periodic intervals.

The various functions in an automobile are often categorized into domains such as the powertrain domain (control of engine and transmission), the chassis domain (control of steering, braking, and suspension), the safety domain (collision warning and mitigation systems), the body domain (control of door locking and unlocking, interior and exterior lighting, windshield washing and wiping, and heating, ventilating, and air conditioning), and infotainment and telematics. Infotainment is the combination of information and entertainment, and includes radios and video entertainment systems, while telematics refers to the combination of telecommunications and computing, such as GMs OnStar system that provides various navigation, concierge, and emergency services.

Because the primary purpose of a vehicle is to provide transportation from one location to another, the most critical systems from a safety and dependability perspective are the control-oriented domains (powertrain, chassis, active safety, and to a lesser extent, body), hence they will be the focus of this survey. The control-oriented domains provide a diagnostic connector used for downloading DTCs and for uploading of software and calibration updates. Due to the criticality of the control-oriented domains, a gateway with a firewall is usually used to isolate them from the infotainment and telematics domains to prevent unauthorized access from public sources.

As mentioned earlier, automotive control systems are highly distributed. Multiple ECUs, typically 20–60 of them in mid-level or premium vehicles, communicate with each other over automotive serial communication protocols such as LIN, CAN, and FlexRay. Local Interconnection Network (LIN) is a slow speed (10 Kbps) master-slave bus used to connect multiple smart sensors or actuators, such as switches and motors (the slave nodes), to a host ECU that initiates and controls the communication schedule (the host node). Controller Area Network (CAN) is a medium-speed (30 Kbps–1 Mbps) peer-to-peer asynchronous bus in which any node may attempt to transmit at any time, and collisions are resolved by a bit-level Carrier Sense Multiple Access / Collision Detection (CSMA/CD) priority scheme. FlexRay is a fast (10 Mbps) synchronous peer-to-peer bus with a time-triggered Time Division Multiple Access (TDMA) access scheme. FlexRay has some similarities, and also some differences, with Time-Triggered Protocol/Class-C (TTP/C). A FlexRay communication cycle may contain both a static segment consisting of statically scheduled time slots, and a dynamic segment consisting of dynamically arbitrated minislots, while a TTP/C TDMA round contains only statically scheduled TDMA slots. Both FlexRay and TTP/C have the option of using either one or two communication channels. If present, the second communication channel may be used either for additional bandwidth or for redundancy. TTP/C includes a membership service as part of the protocol, while FlexRay leaves any membership service to the application layer. Another difference between FlexRay and TTP/C is that FlexRay allows a given node to transmit in more than one slot, while TTP/C restricts a given node to transmitting in only one slot. The static slots in FlexRay must all be configured to have the same duration, while the TDMA slots in TTP/C are allowed to be configured to have different durations. Table 1 provides a high level comparison of the characteristics of LIN, CAN, FlexRay and TTP/C.

A typical automotive architecture may contain multiple LIN,CAN, and FlexRay networks with gateways between them. LIN and CAN always use linear bus topologies, while FlexRay may use either a linear bus or a star topology, or even combinations of the two. A typical generic overall vehicle computing and communication topology is depicted in Figure 1.

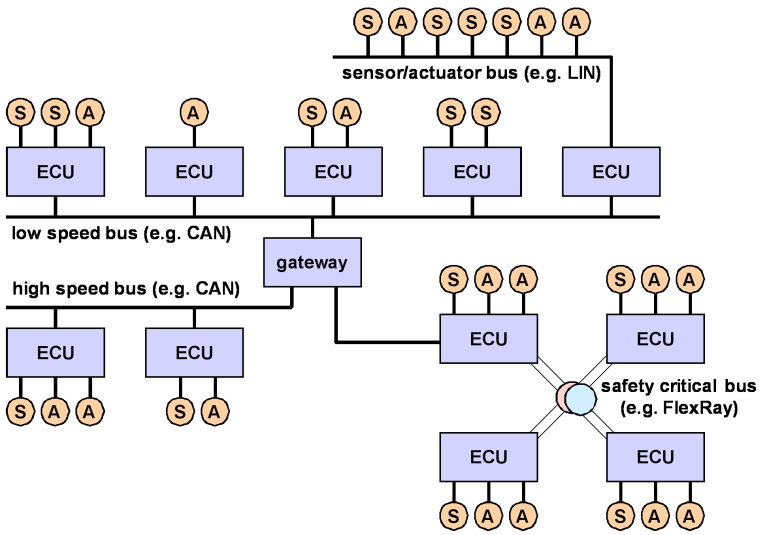

Figure 1: A generic automotive computing and communication topology.

Characteristic	*LIN*	*CAN*	*FlexRay & TTP/C*
Network	master-slave	peer-to-peer (broadcast)	peer-to-peer (broadcast)
Topology	linear bus	linear bus	star or linear bus
Synchronization	synchronous (master-slave)	asyncronous (CSMA/CD)	synchronous (TDMA)
Scheduling	static (design-time)	dynamic (run-time)	static (design-time)
Max Bit Rate	20 Kbps	1Mbps	10 Mbps
Max Msg Payload Size	8 bytes	8 bytes	254 bytes
Redundancy	1 channel	1 channel	1 or 2 channels

Table 1: A high level comparison of the characteristics of LIN, CAN, and FlexRay/TTP.

2.2 Diagnosis & Fault Tolerance

A **failure** is an event that occurs when a system does not behave according its specification. Failures result from faults and errors. A **fault** is simply a latent defect or abnormal condition; a **faulty component** contains such a defect or is subject to such a condition. Faults can cause **errors**, which are anomalies in the internal state of a system. Errors lead to failures when the erroneous state becomes exposed externally as erroneous output. **Error propagation** is the process by which errors are transformed into failures, which can then manifest as faults elsewhere in the system. Containment regions can be used to limit error propagation [33]. Components in a **Fault Containment Region (FCR)** are assumed to be protected from faults external to the region (i.e., a single fault will not directly cause simultaneous errors in two FCRs). However, erroneous data can be propagated from one FCR to another, which results in further faults and errors [33]. An **Error Containment Region (ECR)** prevents this by masking erroneous data. The relationship between faults, errors and failures is captured by the *Fault-Error-Failure* chain [6].

Common-cause and *common-mode* failures are those that can affect multiple units of a fault-tolerant architecture at once. **Common-cause failures** result from a single fault, such as mechanical destruction of

multiple computers, thermal problems, Electromagnetic Interference (EMI), or other types of interference. **Common-mode failures** are caused by multiple, correlated faults that can occur even though the architectural units were designed with diverse design teams and use diverse component types and technologies. This is because design engineers are educated in a similar way even across different universities, and they use similar development processes and tools.

Consider a vehicle that includes a brake-by-wire feature. One perspective might view the ECU and sensors that calculate the position of the brake pedal as a system. In this system, a faulty sensor causes erroneous position values to be calculated by the ECU; the ECU fails when it outputs those erroneous values. Now assume a perspective that views the entire brake-by-wire feature as a system, with each ECU as a discrete component. The failed brake-sensor ECU provides erroneous position values to the brake-controller ECU. The brake-controller ECU uses faulty input to calculate erroneous brake pressure values, and a failure occurs when the brakes apply an incorrect pressure.

As a technique for fault tolerance, **diagnosis** determines the type and location of a fault that led to an error [6]. This implies an investigation with two thrusts. **Localization** identifies the faulty component. **Discrimination** identifies the type of fault(s) present, with respect to a given fault model (e.g., transient vs. intermittent faults, internal vs. external faults, hardware vs. software faults, etc). Each thrust can be associated with a different aspect of fault tolerance. Localization supports the *isolation* of the faulty component so that its effect(s) on the system can be contained. Discrimination can inform the selection of an appropriate *recovery* action based on the fault type. Of course, these associations are not orthogonal and are, in fact, often mutually supportive. For example, after localizing a faulty component, it could be selectively isolated based on whether the identified fault is transient or permanent.

Another related concept is **root-cause analysis**, which identifies the specific defect(s) or condition(s) ultimately responsible for a failure. Root-cause analysis can be distinguished from diagnosis in that root-cause analysis reaches for the ultimate origin of the *Fault-Error-Failure* chain, while diagnosis typically has a more limited scope (e.g., localization to a specific FCR). This makes root-cause analysis valuable for fault prevention and removal, which takes place during the development and maintenance phases of the system's life-cycle. However, because our work is mostly concerned with run-time fault-tolerance techniques, the remainder of this paper will focus on diagnosis.

2.3 Characterizing Faults

Any approach to safety or fault tolerance must begin with a categorization of the faults to be detected and handled. Such a classification is generally referred to as the *fault model*, *fault taxonomy* or *fault hypothesis*. These terms are often used interchangeably, which can result in confusion among researchers with different backgrounds. The purpose of this section is to distinguish and clarify these terms.

A **fault taxonomy** provides a basis for classifying fault types, usually with a hierarchical structure. A comprehensive fault taxonomy that emerged from decades of research in dependable computing [5, 35, 36] is summarized in [6]. Faults are classified according to their *phase of creation* (development vs. operational), *system boundary* (internal vs. external), *cause* (natural vs. human-made), *dimension* (hardware vs. software), *objective* (malicious vs. non-malicious), *intent* (deliberate vs. non-deliberate), *capability* (accidental vs. incompetence), and *persistence* (permanent vs. transient).

Starting with this application-independent taxonomy, a more specific classification must be derived for a particular domain or industry that enumerates the faults particular to their class of systems. For example, a description of fault types for integrated safety systems within the automobile industry was recently created by the Electronic Architecture and System Engineering for Integrated Safety Systems (EASIS) Consortium [19]. The broad categories of faults in the EASIS taxonomy are *CPU faults*, *sensor faults*, *actuator faults*, *power supply faults* and *communication system faults*. Within each broad category, detailed lists of specific faults are enumerated. For example, the *CPU faults* category includes *calculation errors*, *value errors*,

program flow errors, *interrupt errors*, *timing errors*, *RAM/ROM cell errors*, *RAM/ROM addressing errors*, *I/O interface circuit errors*, and *scheduling errors*.

The preceding taxonomies facilitate discussion of which faults should be avoided in a particular system through *prevention* and/or *removal*. However, developing specific mechanisms to tolerate faults through *detection* and *recovery* requires knowing how the faults affect the system in question. Classification based on the behavior that faulty components exhibit upon failure (i.e., the **failure semantics** [16]) provides such information. *Weak semantics* place fewer restrictions on the possible behaviors than *strong semantics*, so more robust – and expensive – mechanisms are required to tolerate them. The strongest class includes components that halt permanently and no longer produce any output (*crash* failures). *Omission* failures occur when a component fails to perform a task, but otherwise continues operating. A component that exhibits *timing* failures produces correct results, but delivers them outside of a specified time window. On the other hand, a component that exhibits *value* failures produces incorrect results. *Byzantine* components may exhibit arbitrary behavior, representing the weakest failure semantics.

A related way to classify faults is according to their detectability [38] and communication patterns [55, 59]. The former grouping [38] classifies faults as malicious or benign. *Benign* faults are locally detectable by all non-faulty components, while *malicious* faults are not. Malicious faults can be either *symmetric* or *asymmetric*, depending on whether all correct nodes receive the same faulty output [55]. In this case, benign faults represent the strongest semantic class, while asymmetric-malicious faults are equivalent to the weakest (i.e., Byzantine) fault class.

A **fault model** captures information that allows faults to be reasoned about in a systematic way. In dependability and fault-tolerance literature, the fault model is generally given by a **fault hypothesis** that specifies assumptions on the number and types of faults that a given system is expected to handle [32]. For example, the EASIS Consortium makes two key assumptions [19]. First, it is assumed that the system is fault-free at the time of initial delivery (i.e., most functional or design faults have already been removed from the system). Second, it is assumed that only one independent fault occurs at a time. This is often called the single-fault assumption, and is a common assumption used in the automobile industry. Fault models that assume all faults belong to a strong (e.g., benign) fault class risk being overly optimistic. On the other hand, pessimistically assuming strictly Byzantine faults can lead to unnecessary expense. Hybrid fault-models, such as the Customizable Fault/Error Model (CFEM) [59], aim to be more flexible by enabling assumptions to be made based on combinations of fault types.

In the field of automotive diagnostics and health management, the fault model is given by a mapping between the known failure modes in a system and the available tests / symptoms. This mapping is often represented in a Dependency Matrix (D-MATRIX) (see Figure 2). Such a fault model can be used to identify the fault source, as well as aid in the study of the diagnostic coverage of various vehicle subsystems. Using the fault model, ambiguity groups and test point placements can be optimally identified for expanding fault coverage.

This type of fault model is the heart of an Integrated Vehicle Health Management (IVHM) system. Fault reasoners that operate on the fault model can be used for rapid fault detection onboard as well as for the generation of diagnostic strategies for helping technicians off board. The fault model is also conceived as a valuable tool for studying existing diagnostic strategies in terms of coverage. Further improving of the diagnosis strategy may be required by reducing ambiguity groups during the design stage. This can be achieved by the addition of supplemental tests as needed to improve fault coverage. For a more detailed discussion of fault models as they relate to reasoners, see [49, 50, 51].

A *maintenance-oriented fault model* proposed by [41] consists of a taxonomy based on hardware *component* and software *job* boundaries, along with assumptions that specify a maintenance action for each class. *Internal* faults originate within a boundary, while *external* faults originate outside of a boundary. A third class, *borderline*, is introduced to capture faults that arise on the boundary itself (e.g., connectors that interface components with the communication network). Faults are first classified according to compo-

Failure modes \ Symptoms	T1	T2	T3	T4	T5	T6	T7	T8	T9
C1			•			•			•
C2				•				•	
C3	•								
C4	•			•					
C5					•				
C6							•		•
C7	•						•		
C8		•		•					
C9						•	•		
C10									•

Table 2: An example Dependency Matrix mapping symptoms to failure modes.

nent boundary (i.e., *component-internal*, *component-external*, and *component-borderline*). Given that jobs run within the component boundary, component-internal faults can be further classified according to the job boundary. These sub-categories are *job-inherent*, *job-external* and *job-borderline*. Component-internal faults are assumed to be permanent, so replacement of either software or hardware is necessary. Job-inherent and job-borderline faults require software updates, while job-external faults require replacement of the hardware component. Component-borderline faults are also assumed to be permanent, and require inspection or replacement of connectors. Component-external faults are assumed to be transient, so no maintenance action is required.

3 Perspectives on Failure Diagnosis

The need for failure diagnosis can be seen from two perspectives: a *maintenance-oriented* perspective and a *safety-oriented* perspective. The maintenance-oriented perspective is concerned with improving diagnosis to aid service personnel and reduce warranty costs (see Section 3.1), in the field, often with the vehicle in an offline mode. The safety-oriented perspective is concerned with supporting the overall dependability of safety-critical systems (see Section 3.2), typically for the vehicle in an online mode.

3.1 Supporting Maintenance and Reducing the "No Fault Found" Phenomenon

Warranty Week reports that warranty costs for automotive manufacturers can be prohibitively high and can arise from multiple issues, such as faulty parts, invalid claims, customer dissatisfaction, part replacements, etc. For just the "big three" automakers (General Motors, Ford and DaimlerChrysler), the total warranty claims topped $15.34 billion, on sales of some 20.8 million units in 2006 [61]. The most pernicious warranty costs arise when parts are replaced to keep a vehicle operational, even though the root cause of the problem has not actually been found and the replaced parts might not actually be to blame.

The term No Fault Found (NFF) refers to cases where a component is replaced, but upon subsequent testing, the faulty condition leading to its replacement cannot be reproduced[1]. NFF incidents are very real in industry. For example, studies report that for a single automotive component (an electronic ignition module),

[1]This phenomenon might also be referred to in the literature as Trouble Not Identified (TNI), No Trouble Found (NTF), Cannot Duplicate (CND), etc.

the number of replaced parts labeled as NFF varied from 25-80% over 4 model years [56]. This phenomenon is not confined to the automotive industry, with statistics showing that NFF rates are fairly significant in the aerospace industry as well [39]. About 50% of units replaced during the utilization and support stages in the aerospace industry were classified as NFF [11, 28, 54]. The Air Transport Association identified more than 120 units with NFF rates in excess of 40% [11].

Automotive control systems are instrumented to produce DTCs that indicate errors and identify the faulty subsystem, component or circuit. DTCs are set when internal error checking routines on individual ECUs detect a faulty condition. DTCs can then be read out of the vehicle by service technicians using special diagnostic tools. However, as the complexity of automotive systems has increased, DTCs and available diagnostic tools have become less able to pinpoint the exact component responsible for a failure [20, 64]. Lacking accurate diagnostic information and under time pressure to resolve problems quickly, service technicians often resort to a "shotgun" troubleshooting approach, where multiple parts are replaced with the hope that one of them will fix the problem [53]. At times, they may even eschew existing diagnostic tools altogether, instead relying on instinct and trial-and-error [56]. In the worst case, faulty parts can remain in the vehicle undiagnosed. In the best case, functional components will be replaced needlessly.

The wasteful replacement of functional components is one cause of high NFF rates and their associated warranty costs. Another cause is laboratory testing that does not accurately reflect real-world operating conditions [56, 62]. In such cases, returned components might actually be faulty, yet appear to operate correctly when removed from the field to undergo laboratory testing. Manufacturers are often forced to assume that field returns constitute real failures since there is no evidence to the contrary [56]. Because brand loyalty and responsive customer service are important, car manufacturers pay the associated warranty and part replacement costs, even in the NFF cases.

Failure diagnosis research could reduce the incidence of NFFs by discovering the true root-cause of problems. This would lead to more accurate diagnostic tools, allowing more informed part replacements and lowering warranty costs. Continual, online diagnosis could further reduce the incidence of NFFs by identifying faults under actual operating conditions.

3.2 Supporting Run-Time Dependability in Safety Critical Systems

As mentioned earlier, automotive embedded control systems are being used to control increasingly critical systems. Considering the increasing levels of control authority and automation, a failure of any component in the system can have great impact on the stability of the motion of the vehicle.

For example, a brake-by-wire control system sends braking torque commands to an electromechanical brake actuator in response to the driver pushing on the brake pedal. There is no mechanical or hydraulic connection between the driver's brake pedal and the brake calipers that squeeze the brake rotor to provide braking torque. The path from the brake pedal to the brake calipers is entirely electrical and electronic, consisting of a brake pedal position or force sensor connected to an ECU, which sends commands to the electromechanical actuator. A fault in any element of this system must be detected and mitigated to avoid loss of braking functionality. The mitigating action to be taken must provide some degree of braking performance, whether full or degraded, and the specific action to be taken may depend on accurate diagnosis of the fault that occurred.

Three important factors identified by [22] that are critical for achieving high dependability of vehicle motion control systems are (1) high fault detection coverage, (2) rapid recovery from failures, or short failure detection latency, and (3) and short time-to-repair periods. For the brake-by-wire example mentioned above, any fault in the sensor, the computer, the actuator, or the wires or communication paths between them, must be detected and diagnosed with high coverage so that the mitigating action can be taken with high reliability, and the time available to perform this failure detection and diagnosis is very limited so that the braking function can be restored as quickly as possible. Existing methods for diagnosing failures are often too broad

to allow for quick, accurate diagnosis [64]. More accurate diagnosis could also help improve dependability with respect to membership protocols [52].

Highly accurate and dependable diagnostic mechanisms with high detection coverage and low detection latency will ensure the safety of the vehicle at run time, and could even slow the rise in complexity by reducing the need for physically redundant components to maintain the safety of the vehicle [63]. It has been proposed that a highly robust diagnosis framework could be used to ensure the safety of systems that are composed of less robust subsystems [14].

Within the context of diagnosis, the violation of fail-silence assumptions can be seen as examples of non-faulty nodes being misdiagnosed by a fault tolerance mechanism. Fault-injection experiments using heavy-ion fault injection have shown fail-silence violations [52] and error propagation [1] in TTP/C. Fault-injection experiments using hardware models have show that transient faults in the CAN Communication Controller (CC) and Communication Network Interface (CNI) can result in masquerade failures, where a faulty node "impersonates" a non-faulty node [47]. Experiments performed using modeled FlexRay controllers have highlighted instances of error propagation in FlexRay bus and star topologies [17].

4 Existing Approaches to Failure Diagnosis

4.1 Model-based diagnosis

Control algorithm designers typically develop control algorithms that can tolerate faults in sensors, actuators, and in the physical plant being controlled, but they normally assume that the computing hardware upon which their control algorithms execute, and the serial communication links with which they communicate, don't fail. Furthermore, they normally assume that their control software itself is correct.

Classical automotive diagnosis, at least for continuous systems, is usually based on a control-theoretic approach. In this approach, the control system to be diagnosed is modeled based on the physics of the system, and this model of the system is executed at run-time along with the actual system. Control-theoretic state observers are used to estimate state variables that cannot be measured directly. At run time, the actual outputs of the control system are compared with the outputs predicted by the model. Residual values are calculated as the differences between the actual and predicted outputs. Threshold values of the various residuals, or of combinations of residuals, are correlated at design time with various anticipated faults from the fault model. During run time, whenever such residual values are encountered that exceed predefined thresholds, the corresponding fault is considered to have occurred.

The appeal of the analytical redundancy (model-based) approach lies in the fact that the existing redundancy can simply be evaluated by information processing, without the need of additional instrumentation in the physical system. However, there is a price to pay for this benefit, which arises from the need for a mathematical model for the system. . The weakness of this approach stems from the additional computational expenditure that is required for on-line modeling and from the sensitivity of the diagnosis system to modeling error that is not avoidable in practice. Several model-based diagnosis methods have been developed [3, 24, 18, 46, 45]. The general approach of developing model-based diagnosis consists of the detection of faults in the processes, actuators and sensors by using the relationship and the dependencies between different measurable signals. These relationships are expressed by mathematical process models. Figure 2 shows the basic structure of model-based fault diagnosis process. Based on measured input signals U and output signals Y, the diagnosis methods generate residuals r, parameter estimates Θ or state estimates $\hat{x}$, which are called features. By comparison with the normal features, changes of features are detected, leading to analytical symptoms s. Model-diagnosis use state and parameter estimation methods such as Kalman Filter, Particle Filter, and Observers. Parameter estimation methods are also used as part of the model-based equations. For a more detailed discussion of model-based diagnosis methods, see [27].

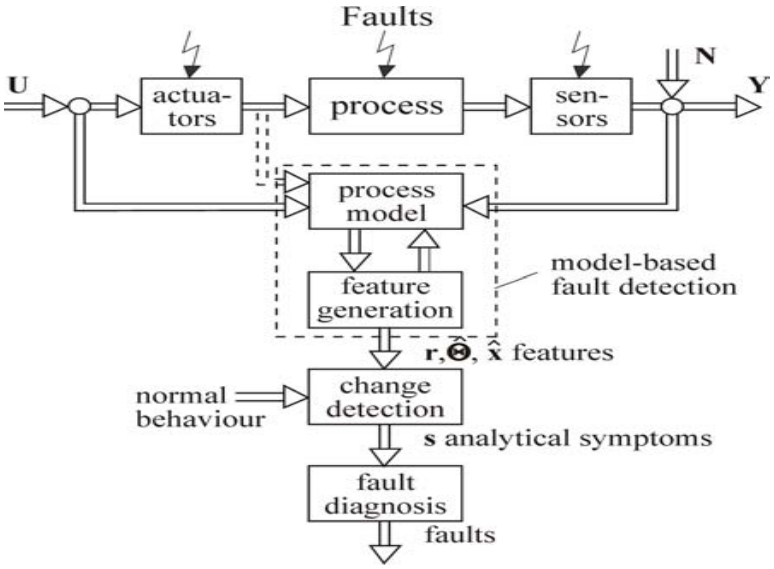

Figure 2: The general process for model-based diagnosis [27].

The control-theoretic approach has been used in powertrain control systems [37] and in chassis systems such as brake control systems [25]. The fault model typically associated with the control-theoretic approach includes sensor faults, actuator faults, and faults in the mechanical, electromechanical, or hydraulic plant being controlled. Conspicuously absent from the fault model are the computer hardware, software, memory devices, and communication links used in the control system itself – these are assumed to be always operating in their fault-free modes.

4.2 Distributed embedded systems

Diagnosis can also be approached from the system level. The classic formulation of system diagnosis is the *Preperata-Metze-Chien (PMC) model* [44]. Under the PMC model, components test each other according to predetermined assignments. The set of test results (called the **syndrome**) can then be collected and analyzed to determine the status (faulty or fault-free) of each component. This ultimately allows a consistent view of the system state to be achieved in the presence of faults. Byzantine agreement [34] has a similar goal. While the general problem – reaching consensus in distributed systems – is the same, the strategy used by these two perspectives is different [9]. Byzantine agreement aims to mask faults without regard to which node is actually faulty, while system diagnosis is concerned with identifying the faulty node so that it can be isolated.

The original PMC model used assumptions that made it impractical for application in real fault-tolerant systems [9]. Many of these assumptions have specific implications for the automotive domain.

Scheduling. Diagnostic task scheduling in the PMC model is agnostic toward non-diagnostic tasks. However, the primary function of a system is not typically to diagnose itself; it is to perform a service, such as applying the brakes when the brake pedal is depressed. The resources needed to run diagnostic tasks, including testing and analysis, therefore have to be considered within the context of the functional tasks.

Fault persistence. Only permanent faults are considered by the PMC model. However, intermittent faults (which often portend permanent faults) and transient faults are common in automotive systems as well. The action taken on the diagnostic output may differ depending on the persistence of the fault.

Centralized analysis. Implicit in the PMC model is the idea that syndromes are collected and analyzed at a centralized location [9], which represents a single point-of-failure. Such single points of failure must be avoided in fault tolerant architectures.

Perfect tests. Another unrealistic assumption of the PMC model is that tests produce pass/fail answeres with 100% coverage and accuracy. In practice, perfect coverage is very difficult to achieve.

Subsequent work has extended the PMC model by addressing many of these limitations. An extensive survey of such work has already been presented in [9]. Rather than duplicate their already broad treatment, we will focus on automotive-specific examples in depth.

Online diagnosis algorithms were introduced by [58] that operate under the CFEM [59]. The goal of these algorithms is to support runtime isolation and recovery. They assume deterministic, frame-based, broadcast communication between nodes. Instead of scheduling explicit tests, these algorithms use the correctness of existing messages as test stimuli. In a sense, each component performs an implicit test on every component that it communicates with. Individual nodes collect local syndromes based on the correctness of messages that they have received from other nodes. In order to diagnose locally undetectable faults and reach a globally consistent diagnosis, these local syndromes are then exchanged between nodes using an agreement algorithm. The severity of faults is addressed using penalty-counts that are weighted by fault-detection method. Transient faults can be handled by updating the penalty count over time according the *decay rate*, which is based on the time that the errors caused by a single fault are expected to remain in the system.

The online algorithms developed by [58] were extended to discriminate healthy nodes from unhealthy nodes in time-triggered automotive systems [48]. A healthy node is one that exhibits only external transient faults, while an unhealthy node exhibits intermittent or permanent faults. The basic idea is that faulty messages that are locally detectable by at least one receiver (i.e., benign and asymmetric faulty messages) can be analyzed over time to discriminate healthy nodes from unhealthy nodes. Diagnosis proceeds as follows. Each node broadcasts its *local syndrome* using a *diagnostic message*. The local syndrome itself is created by observing the validity bits of diagnostic messages sent by other nodes. Time is not explicitly included in the diagnostic message. However, the periodic sending of diagnostic messages implicitly encodes the temporal relationship between faults. Each node aggregates diagnostic messages into a *diagnostic matrix* that relates a node's opinion about its peers in rows (e.g., the node's local syndrome) to the group's opinion about a node (e.g., the node's validity bit in all other local syndromes) in columns. The columns are voted into a single *consistent health vector*, which is then analyzed using a penalty / reward algorithm. The penalty / reward algorithm uses two pairs of counters and thresholds that determine when a node is faulty. The penalty counter is increased when a node's entry in the consistent health vector indicates a fault, otherwise the reward counter is increased according to the *criticality* of the node. When the reward threshold for a node is crossed, the penalty counter for that node is reset to zero. When the penalty threshold for a node is crossed, the node is diagnosed as faulty. Reintegration is not addressed. The penalty and reward thresholds, as well as the criticality values, are tunable in order to allow tradeoffs to be made between availability and diagnostic latency. A prototype implementation of the protocol is built using TTP/C controllers and analyzed via physical fault injection.

Online algorithms have also been proposed to diagnose faulty actuators in distributed control systems [29]. The algorithms assume reliable, deterministic, broadcast communication between system components (i.e., a time-triggered bus). The general idea is to leverage existing actuation commands as test stimuli, avoiding the need to schedule dedicated tests. Sensors built into an actuator monitor its physical operation after a command is received. Based on the sensed value, an estimate of the original command can be derived from a model. If diverse sensors and/or models are available, a single estimate is arrived at using approximate agreement. The actual and derived commands are compared, and a failure is diagnosed if the difference

exceeds some threshold. A key advantage of this approach is that it provides an opportunity to replace physical redundancy (i.e., using multiple actuators to mask a failed actuator) with analytical redundancy (i.e, using diverse sensors / models to diagnose and isolate a failed actuator). Of course, this assumes that appropriate sensors and models are available for a given actuator. Diagnostic tasks (reading sensors, deriving estimates, etc) are *distributed* among multiple processors, and a global view of the diagnosis is reached using agreement. Distributing the diagnostic tasks allows the algorithm to tolerate faults in processors and sensors. Note, though, that the algorithm does not *diagnose* faults in processors or sensors. Task scheduling algorithms take into account the *control period* required for acceptable application response-times as well as the *diagnostic latency* required to keep the system from reaching an unsafe state following a failure.

Out-of-norm Assertions (ONAs) are introduced as a way to correlate fault effects in the three dimensions of value, time and space [42]. The ONA mechanism is designed to operate under a maintenance-oriented fault model (see Section 2.3). Faults are identified by monitoring selected variables of a component's interface state for deviations from normal operation. Specifically, the set of monitored variables and the conditions under which they are said to deviate is a *symptom* of a fault. Fault-induced changes in the value, time and space domain of the distributed state are described by *fault patterns*. The *distributed state* is the combined interface state of all of the system components. ONAs operate on the distributed state by encoding symptoms from multiple components. A fault is diagnosed when when all of the symptoms of a particular fault-pattern are present. The ONA mechanism underlies a framework for diagnosing failures in time-triggered networks [40]. A prototype implementation of the framework using TTP/C controllers instruments the frame status field of the controller. When the frame status field indicates an error, the specific error is encoded as a symptom. The symptom (value domain) is combined with a global synchronized timestamp (time domain) and the unique identifier of the component (space domain) to form a diagnostic message. Diagnostic messages are sent to a dedicated diagnostic subsystem for analysis using a simple threshold-based algorithm [13]. Although the diagnostic subsystem itself is centralized, its individual tasks can be distributed across multiple processors using by the underlying architecture. In order for the processors to maintain a consistent view of the diagnosis, it would need to be disseminated back to them from the diagnostic subsystem. The framework is applied to connector faults in [43].

A platform- and application-agnostic approach demonstrates the extent to which diagnosis is possible using only passive monitoring in FlexRay-based networks [4]. The approach is constrained by purposefully restrictive goals. Specifically, a dedicated diagnostic node is given access only to the communication bus, the diagnostic node cannot disrupt the normal operation of the system, and no modifications (system-level, application-level or otherwise) can be made to existing nodes or architectures. The authors identify several aspects of the FlexRay protocol that can be used to aid diagnosis under such restrictions, such as syntactic failures in the value domain (e.g., Cyclic Redundancy Check (CRC) mismatches), semantic failures in the value domain (e.g., application specific plausibility checks) and failures in the time domain (e.g., early, late or missing messages). The authors suggest that this information is sufficient for concluding that a node is able to transmit messages (i.e., diagnosing faults in the "transmit path" of a communication controller. However, more information is needed to conclude that a node is able to receive messages (i.e., to diagnose faults in the "receive path"). The authors introduce a method for obtaining this information by perturbing the FlexRay clock-synchronization protocol. By sending specially crafted synchronization frames, the diagnostic node can cause other nodes to synchronize to a communication cycle that is slightly off from the configured cycle length. Keeping the modified cycle length within a certain tolerance of the configured cycle length allows the network provide normal services to the application. Nodes that do not synchronize to the modified cycle are assumed to have not received the clock synchronization frames sent by the diagnostic node, indicating a fault in their receive path. Furthermore, faulty synchronization messages can be injected by the diagnostic node to probe the error checking facilities of the nodes under test. Nodes that do not reject the faulty messages will remain synchronized to the modified cycle length, while nodes that do correctly reject them will begin drifting back to the configured cycle length. This approach does not require active

participation in the diagnostic process by the components being diagnosed. However, this implies a centralized diagnosis, which does not provide a globally consistent view of faulty components. This approach is best suited to testing in a maintenance environment, as opposed to runtime diagnosis while the vehicle is on the road.

A component-based diagnostic strategy breaks vehicle-level diagnostics down into a hierarchy of systems and subsystems [63]. Each level in the hierarchy is made up of modules. Modules at the bottom of the hierarchy are individual system components. Modules in the middle are subsystems made up of multiple components. The highest level represents whole-vehicle diagnosis. A supervisory strategy is employed between modules on different levels, but no direct diagnostic interaction is allowed between modules on the same level. Furthermore, a module can only be supervised by a single higher-level module (i.e., a component can't belong to more than a single diagnostic subsystem). Each diagnostic module can have seven diagnostic outputs (*normal*, *need maintenance (no danger)*, *need maintenance (danger)*, *intermittent fault*, *no redundancy*, *fail-silent*, *failure*) that are fed to supervisory modules for additional decision making.

4.3 Enterprise systems

While discussing failure-diagnosis issues for embedded automotive systems, it is important and relevant to examine how enterprise systems handle these issues. Enterprise systems are typically not safety-critical; however, they have fairly demanding requirements in terms of availability because they are required to be operational 24×7, often with disastrous consequences (bad publicity, loss of revenue, etc.) when downtime occurs [26]. Enterprise systems are also fairly large, covering multiple geographic regions, thousands of computers and significant amounts of network traffic—their diagnostic procedures must often be autonomous because it is often infeasible to require manual intervention for such large, complex systems.

Fault detection approaches in enterprise systems, such as e-Commerce systems, typically localize anomalous system behavior through statistical analysis of time-series data, or through dependency analysis of request flows. The data sources used in time-series analysis range from resource usage metrics, to sequences of events in system-call and error logs. [15] use machine learning to identify the resource usage metrics that are most correlated with performance problems. They generate a database problem signatures which they use to diagnose recurrent problems. [12, 23] profile resource usage across the multiple layers of the system, and develop rules-of-thumb to common system problems. [21, 60] detect problems by identifying for anomalous sequences of system calls.

An alternative approach to fault-detection in enterprise systems analyzes dependencies in request flows to detect either high-latency communication paths, or unexpected structural changes in paths due to exceptions or process crashes. [2] isolate performance problems by passively monitoring messages exchanges across components, inferring causal message flows, and identifying high-latency paths. [31] monitor causal flows in Java-based applications to detect changes in the shapes of the path. Sherlock models the probability of error propagation on communication paths, and applies Bayesian analysis to infer the source of the problem [7].

Hybrid approaches extract both resource usage and request flow information from a distributed system, and attribute resource consumption to individual requests to construct concise workload models which can be used for anomaly detection [10].

Gumshoe attempts to diagnose performance problems in replicated file systems [30]. Various Operating System (OS) and protocol-level metrics are gathered and then analyzed using peer-comparison. Gumshoe was able to effectively localize faulty nodes under two different replicated file systems.

Fault detection approaches developed for enterprise systems might not be directly applicable to automotive systems because automotive systems have limited processing and storage capacity and might not support the level of instrumentation and processing needed by the enterprise approach. Automotive systems also require a higher degree of accuracy due to the safety-critical nature of chassis and powertrain

subsystems. More interestingly, though, there are fundamental differences between enterprise and embedded systems that we must take into account when attempting to apply any enterprise diagnostic practices to autonomous driving systems.

For one, enterprise systems are transactional and event-triggered in nature, which means that they usually focus on preserving data and tend to center around end-to-end request-response semantics. Embedded systems are continuous-control and time-triggered. They tend to employ clock synchronization algorithms, consist of operations with timing guarantees, and involve real-time data and processing. Enterprise systems tend to be resource-intensive, involving high bandwidth non-real time Internet Protocol (IP)-based data networks, with more than adequate resources (in terms of disk, memory, Central Processing Unit (CPU), battery power); enterprise desktop systems typically use full-featured non-real-time commercial off-the-shelf OSes, with virtual memory and memory protection capabilities. Embedded systems often work with severe resource constraints (limited bandwidth non-IP-based real-time control network, limited memory / CPU, often no disk, battery powered); furthermore, features such as memory protection and virtual memory are often disabled, and the OSes tend to be customized (rather than off-the-shelf) with a reduced feature-set and real-time scheduling capabilities. All of these enterprise vs. embedded differences make it difficult, if not impossible, to directly apply enterprise diagnostic techniques to autonomous driving systems.

Nevertheless, the general architectural aspects of instrumentation, data-analysis, and dependency-tracing are equally applicable to both enterprise and embedded automotive systems.

5 Summary

Various fault taxonomies, fault hypotheses, and fault models have been proposed for use in the automobile industry. One example is the fault taxonomy proposed by the EASIS consortium [19] that includes CPU faults, sensor faults, actuator faults, power supply faults, and communication system faults. EASIS also proposes a fault hypothesis that assumes at most one of the above faults exists at any given time. Maintenance-oriented fault models focus on off-line support for service technicians, while safety-oriented fault models focus on on-line support (run-time architecture) for maintaining safe operation of the vehicle while being driven.

The classic formulation of system-level diagnosis is the PMC model, named after the authors [44] In this graph-theoretic approach, each node tests a subset of the other nodes, and the collection of all such pass-fail test results, called the syndrome, is evaluated using graph algorithms to come up with a system-wide consensus on which nodes are faulty so that they can be isolated. The theory provides proofs of conditions under which the system is diagnosable for a given number of faults. Byzantine agreement protocols [34] have the same goal as the PMC approach, namely, to reach consensus on the state of health of a distributed system, but approach the problem in a different manner. Instead of attempting to identify the faulty nodes so that they can be isolated, Byzantine agreement protocols aim to mask faults by agreeing on a set of good outputs to propagate through the system.

Several diagnostic techniques exist that are unique to time-triggered communication networks such as FlexRay and TTP/C. For example, a method by [48] involves the transmission of diagnostic message consisting of local syndromes, and the aggregation, voting, and analysis of these diagnostic messages to consense on a single system-wide health vector. A method by [4] diagnoses the receive path in FlexRay controllers by perturbing the FlexRay clock synchronization protocol during runtime in a way that does not interfere with the normal operation of the system.

A key trade-off identified by this survey is the distinction between two different approaches for system-level diagnosis of failures in run-time safety architectures. One approach aims to identify and isolate faulty components and to leverage this information to enable more focused and efficient mitigation actions, while the other approach aims to mask failures by consensing on non-faulty outputs to propagate through the system. Future work will further explore this fundamental trade-off.

Appendix: List of Acronyms

CAN	Controller Area Network . . . 3
CC	Communication Controller . . . 9
CFEM	Customizable Fault/Error Model . . . 6
CND	Cannot Duplicate . . . 7
CNI	Communication Network Interface . . . 9
CPU	Central Processing Unit . . . 14
CRC	Cyclic Redundancy Check . . . 12
CSMA/CD	Carrier Sense Multiple Access / Collision Detection . . . 3
DTC	Diagnostic Trouble Code . . . 1
D-MATRIX	Dependency Matrix . . . 6
EASIS	Electronic Architecture and System Engineering for Integrated Safety Systems . . . 5
ECR	Error Containment Region . . . 4
ECU	Electronic Control Unit . . . 2
EMI	Electromagnetic Interference . . . 5
FCR	Fault Containment Region . . . 4
IP	Internet Protocol . . . 14
IVHM	Integrated Vehicle Health Management . . . 6
LIN	Local Interconnection Network . . . 3
NFF	No Fault Found . . . 7
NTF	No Trouble Found . . . 7
ONA	Out-of-norm Assertion . . . 12
OS	Operating System . . . 13
PMC	Preperata-Metze-Chien . . . 10
TDMA	Time Division Multiple Access . . . 3
TNI	Trouble Not Identified . . . 7
TTP/C	Time-Triggered Protocol/Class-C . . . 3

References

[1] Astrit Ademaj, Håkan Sivencrona, Günter Bauer, and Jan Torin. Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proceedings, 2003 International Conference on Dependable Systems and Networks*, DSN '03, pages 123–132, Los Alamitos, CA, USA, June 2003. IEEE Computer Society.

[2] Marcos Kawazoe Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings, 19th ACM Symposium on Operating Systems Principles*, SOSP '03, pages 74–89, New York, NY, USA, October 2003. ACM.

[3] Sanket Amberkar. Diagnostic strategies for advanced automotive systems. SAE Technical Paper Series 2002-21-0024, SAE International, Warrendale, PA, USA, October 2002.

[4] Eric Armengaud and Andreas Steininger. Pushing the limits of online diagnosis in flexray-based networks. In *Proceedings, 2006 IEEE International Workshop on Factory Communication Systems*, WFCS '06, pages 44–53, Piscataway, NJ, USA, June 2006. IEEE.

[5] Algirdas Avižienis. Fault-tolerance: The survival attribute of digital systems. *Proceedings of the IEEE*, 66(10):1109–1125, October 1978.

[6] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January–March 2004.

[7] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. *ACM SIGCOMM Computer Communication Review*, 37(4):13–24, August 2007.

[8] Christopher Baker, David Ferguson, and John Dolan. Robust mission execution for autonomous urban driving. In *Proceedings, 10th International Conference on Intelligent Autonomous Systems*, IAS-10, pages 155–163, Amsterdam, The Netherlands, July 2008. IOS Press.

[9] Michael Barborak, Miroslav Malek, and Anton Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[10] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *Proceedings, 6th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '04, pages 259–272, Berkeley, CA, USA, December 2004. USENIX Association.

[11] Israel Beniaminy and David Joseph. Reducing the "no fault found" problem: Contributions from expert-system methods. In *2002 IEEE Aerospace Conference Proceedings*, volume 6, pages 2971–2973, Piscataway, NJ, USA, March 2002. IEEE.

[12] Sapan Bhatia, Abhishek Kumar, Marc E. Fiuczynski, and Larry L. Peterson. Lightweight, high-resolution monitoring for troubleshooting production systems. In *Proceedings, 8th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '08, pages 103–116, Berkeley, CA, USA, December 2008. USENIX Association.

[13] Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico, and Fabrizio Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers*, 49(3):230–245, March 2000.

[14] Simon P. Brewerton, Frank Grosshauser, and Rolf Schneider. Practical use of autosar in safety critical automotive systems. SAE Technical Paper Series 2009-01-0748, SAE International, Warrendale, PA, USA, April 2009.

[15] Ira Cohen, Steve Zhang, Moisés Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. *ACM SIGOPS Operating Systems Review*, 39(5):105–118, December 2005.

[16] Flavin Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[17] Mehdi Dehbashi, Vahid Lari, Seyed Ghassem Miremadi, and Mohammad Shokrollah-Shirazi. Fault effects in flexray-based networks with hybrid topology. In *Proceedings, 3rd International Conference on Availability, Reliability and Security*, ARES '08, pages 491–496, Los Alamitos, CA, USA, March 2008. IEEE Computer Society.

[18] Eve Limin Ding and Ralf Herbst. Sensor system with monitoring device. US Patent 6625527, Continental Teves AG & Co. OHG, September 2003.

[19] EASIS Consortium. Description of fault types. Deliverable D1.2-13, EASIS Consortium, November 2006.

[20] Thomas Foran and Brendan Jackman. An intelligent diagnostic system for distributed multi-ecu automotive control systems. SAE Technical Paper Series 2005-01-1444, SAE International, Warrendale, PA, USA, April 2005.

[21] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings, 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA, USA, May 1996. IEEE Computer Society.

[22] Robert C. Hammett and Philip S. Babcock. Achieving 10^9 dependability with drive-by-wire systems. SAE Technical Paper Series 2003-01-1290, SAE International, Warrendale, PA, USA, March 2003.

[23] Matthias Hauswirth, Peter F. Sweeney, Amer Diwan, and Michael Hind. Vertical profiling: Understanding the behavior of object-oriented applications. In *Proceedings, 19th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOSPLA '04, pages 251–269, New York, NY, USA, October 2004. ACM.

[24] Hong Xing Hu, Mutasim Salman, Michael Douglas Rizzo, John E. Dickinson, Douglass L. Carson, Eldon Leaphart, and Steven Lee Tracht. Active brake control diagnostic. US Patent 5707117, General Motors Corporation, Warren, MI, USA, July 1998.

[25] Kunsoo Huh, Kwangjin Han, Daegun Hong, Joogon Kim, Hyungjin Kang, and Paljoo Yoon. A model-based fault diagnosis system for electro-hydraulic brake. SAE Technical Paper Series 2008-01-1225, SAE International, Warrendale, PA, USA, April 2008.

[26] IBM Global Services. Improving systems availability. White Paper. http://www.dis.uniroma1.it/ irl/docs/availabilitytutorial.pdf, 1998.

[27] Rolf Isermann. Model-based fault-detection and diagnosis – status and applications. *Annual Reviews in Control*, 29(1):71–85, May 2005.

[28] Ian James, David Lumbard, Ian Willis, and John Goble. Invesigating no fault found in the aerospace industry. In *Proceedings, 2003 Annual Reliability and Maintainability Symposium*, pages 441–446, Piscataway, NJ, USA, January 2003. IEEE.

[29] Nagarajan Kandasamy, John P. Hayes, and Brian T. Murray. Time-constrained failure diagnosis in distributed embedded systems: Application to actuator diagnosis. *IEEE Transactions on Parallel and Distributed Systems*, 16(3):258–270, March 2005.

[30] Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Gumshoe: Diagnosing performance problems in replicated file-systems. In *Proceedings, 2008 IEEE Symposium on Reliable Distributed Systems*, SRDS '08, pages 137–146, Los Alamitos, CA, USA, October 2008. IEEE Computer Society.

[31] Emre Kiciman and Armando Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, 16(5):1027–1041, September 2005.

[32] Hermann Kopetz. On the fault hypothesis for a safety-critical real-time system. *Automotive Software - Connected Services in Mobile Networks*, 4147:31–42, Lecture Notes in Computer Science 2006.

[33] Jaynarayan H. Lala and Richard E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, January 1994.

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, July 1982.

[35] Jean-Claude Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Proceedings, 15th IEEE International Symposium on Fault-Tolerant Computing*, FTCS-15, pages 2–11, Piscataway, NJ, USA, June 1985. IEEE.

[36] Jean-Claude Laprie, Algirdas Avižienis, and Hermann Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[37] Geoffrey McCullough, Neil McDowell, and George Irwin. Fault diagnostics for internal combustion engines – current and future technologies. SAE Technical Paper Series 2007-01-1603, SAE International, Warrendale, PA, USA, April 2007.

[38] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, April 1991.

[39] Michael Pecht and Vijay 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*, 15(6):1160–1164, December 1992.

[40] Philipp Peti and Roman Obermaisser. A diagnostic framework for integrated time-triggered architectures. In *Proceedings, 9th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, ISORC '06, page 11pp, Los Alamitos, CA, USA, April 2006. IEEE Computer Society.

[41] Philipp Peti, Roman Obermaisser, Astrit Ademaj, and Hermann Kopetz. A maintenance-oriented fault model for the DECOS integrated diagnostic architecture. In *Proceedings, 19th IEEE International Parallel and Distributed Processing Symposium*, IPDPS '05, pages 128–136, Los Alamitos, CA, USA, April 2005. IEEE Computer Society.

[42] Philipp Peti, Roman Obermaisser, and Hermann Kopetz. Out-of-norm assertions. In *Proceedings, 11th IEEE Real Time and Embedded Technology and Applications Symposium*, RTAS '05, pages 280–291, Los Alamitos, CA, USA, March 2005. IEEE Computer Society.

[43] Philipp Peti, Roman Obermaisser, and Harald Paulitsch. Investigating connector faults in the time-triggered architecture. In *Proceedings, 11th IEEE Conference on Emerging Technologies and Factory Automation*, ETFA '06, pages 887–896, Piscataway, NJ, USA, September 2006. IEEE.

[44] Franco P. Preperata, Gernot Metze, and Robert T. Chien. On the connection asssignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–854, December 1967.

[45] Pierre-Franç Quet and Mutasim Salman. Model-based sensor fault detection and isolation for x-by-wire vehicles using a fuzzy logic system with fixed membership functions. In *Proceedings, 2007 American Control Conference*, ACC '07, pages 2314–2319, Piscataway, NJ, USA, July 2007. IEEE.

[46] Giorgio Rizzoni, Ahmed Soliman, Pierluigi Pisu, Sanket Amberkar, and Brian T. Murray. Model-based fault detection and isolation system and method. US Patent 6766230, Ohio State University, Delphi Automotive Systems, July 2004.

[47] Hassan Salmani and Seyed Ghassem Miremadi. Contribution of controller area networks controllers to masquerade failures. In *Proceedings, 11th Pacific Rim International Symposium on Dependable Computing*, PRDC '05, page 5, Los Alamitos, CA, USA, December 2005. IEEE Computer Society.

[48] Marco Serafini, Neeraj Suri, Jonny Vinter, Astrit Ademaj, Wolfgang Brandstäter, Fulvio Tagliabò, and Jens Koch. A tunable add-on diagnostic protocol for time-triggered systems. In *Proceedings, 2007 IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, pages 164–174, Los Alamitos, CA, USA, June 2007. IEEE Computer Society.

[49] Mojdeh Shakeri, Krishna R. Pattipati, Vijaya Raghavan, and A. Patterson-Hine. Optimal and near-optimal algorithms for multiple fault diagnosis with unreliable tests. *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews*, 28(3):431–440, August 1998.

[50] John W. Sheppard and Stephyn G. W. Butcher. A formal analysis of fault diagnosis with d-matrices. *Journal of Electronic Testing*, 23(4):309–322, August 2007.

[51] John W. Sheppard, Mark A. Kaufman, and Timothy J. Wilmering. IEEE standards for prognostics and health management. In *2008 IEEE AUTOTESTCON Proceedings*, pages 97–103, Piscataway, NJ, USA, September 2008. IEEE.

[52] Håkan Sivencrona, Per Johannessen, and Jan Torin. Protocol membership in dependable distributed communication systems – a question of brittleness. SAE Technical Paper Series 2993-01-0108, SAE International, Warrendale, PA, USA, March 2003.

[53] Peter Söderholm. A system view of the no fault found (NFF) phenomenon. *Reliability Engineering & System Safety*, 92(1):1–14, January 2007.

[54] Michael D. Sudolsky. The fault recording and reporting method. In *1998 IEEE AUTOTESTCON Proceedings*, pages 429–437, Piscataway, NJ, USA, August 1998. IEEE.

[55] Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *Proceedings, 7th IEEE Symposium on Reliable Distributed Systems*, SRDS '88, pages 93–100, Los Alamitos, CA, USA, October 1988. IEEE Computer Society.

[56] Dawn A. Thomas, Ken Ayers, and Michael Pecht. The "trouble not identified" phenomenon in automotive electronics. *Microelectronics Reliability*, 42(4–5):641–651, April/May 2002.

[57] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Red Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha

Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, July 2008.

[58] Chris J. Walter, Patrick Lincoln, and Neeraj Suri. Formally verified on-line diagnosis. *IEEE Transactions on Software Engineering*, 23(11):684–721, November 1997.

[59] Chris J. Walter and Neeraj Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 290(2):1223–1251, January 2003.

[60] Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. Automatic misconfiguration troubleshooting with peerpressure. In *Proceedings, 6th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '04, pages 245–258, Berkeley, CA, USA, December 2004. USENIX Association.

[61] Warranty Week. Auto warranty vs. quality. http://www.warrantyweek.com/archive/ww20060620.html, June 2006.

[62] R. Williams, J. Banner, I. Knowles, M. Dube, M. Natishan, and M. Pecht. An investigation of "cannot duplicate" failures. *Quality and Reliability Engineering International*, 14(5):331–337, September/October 1998.

[63] Song You and Laci Jalics. Hierarchical component-based fault diagnostics for by-wire systems. SAE Technical Paper Series 2004-01-0285, SAE International, Warrendale, PA, USA, March 2004.

[64] Song You, Mark Krage, and Laci Jalics. Overview of remote diagnosis and maintenance for automotive systems. SAE Technical Paper Series 2005-01-1428, SAE International, Warrendale, PA, USA, April 2005.