Secure Information Sharing in Mixed–Criticality Systems

Armin Wasicek, Thomas Mair

***Abstract*—In this paper we discuss the application of integrity models in a mixed–criticality system to enable the secure sharing of information. The sharing of resources and information enables cost savings. It is the main driving factor in integrated architectures which are used to implement mixed–criticality real-time systems. The major challenge of these systems is simple: low criticality applications must be prevented from interfering with high criticality ones which execute in the same system. An example for such an integrated architecture is the the ACROSS MPSoC architecture which facilitates the implementation of hard real–time systems. We present an integrity model for the secure exchange of information between different levels of criticality within ACROSS. Our approach is based on Totel's integrity model which proposes to upgrade information from low to high by rigorously validating this information. We use different anomaly detection algorithms for these validation objects. Furthermore, we developed an automotive case study for the hardware–in–the–loop simulation of ABS and odometer subsystems in order to provide a proof–of–concept implementation and to evaluate these algorithms within an automotive case study. We were able to show that the encapsulation mechanisms of the ACROSS architecture support the implementation of the proposed integrity model. Moreover, all of the selected anomaly detection algorithms validate the information flow correctly with respect to the defined integrity model. For some of these algorithms, we are able to propose tuning parameters. Summarizing, we are able to show that the secure sharing of information is feasible in a mixed–criticality system. Integrating several subsystems in a single Multi–Processor System–on–a–Chip (MPSoC) not only reduces the number of required hardware units but also enables new ways to implement services.**

***Index Terms*—Mixed–criticality, security, MPSoC, Totel's integrity model**

I. Introduction

Mixed–criticality systems integrate applications with different levels of safety and security in a single computer system. The challenge in mixed–criticality systems is to prevent faults and intrusions that propagate from applications with lower criticality levels to applications having a higher criticality level. Therefore, low criticality applications are usually prohibited from communicating to ones having a higher criticality. The rules defining these communication flows are called integrity models and they guarantee a proper way of communication in the safety and security domains.

One solution is to build all software at the highest criticality level. This does not only increase complexity, but system development also becomes very expensive. For example, a high criticality application might want to read a sensor. Using sensors with the same (high) criticality level will most likely cause a higher cost than using sensors with a lower criticality level. Methods have to be researched to facilitate the safe and secure sharing of information between criticality levels and to enable the simple and cost–efficient implementation of a mixed–criticality system.

In real–time systems, upgrading data from lower level to be usable in higher levels requires *maintaining consistency in time and space* between different data sets.

The research on the theoretic background of integrity models reaches back to the seventies. Models like Bell–LaPadula and Biba are taught in undergraduate security courses. These models define rules how information may be exchanged between criticality levels. However, in many cases these rules are too restrictive. A more recent model is Totel's model, which provides a formal foundation to enable a more flexible information sharing.

Our approach encompasses the specification and the implementation of a secure information sharing system based on Totel's model. We explicitly address real–time constraints by building on the ACROSS architecture which implements a time–triggered System–on–a–Chip (SoC) for application in real–time systems. Our main contributions center around the design of a *Validation Middleware* to check and upgrade information:

- Implementation of a secure information sharing system within the context of the ACROSS MPSoC architecture
- Analysis of different anomaly detection algorithms employed to recognize modifications and voting strategies to upgrade information
- Evaluation of the system in a vehicular simulator

The remainder of the paper is organized as follows: The next Section II introduces the basic concepts and summarizes the related work. In Section III, we introduce the theoretical background, and in Section IV we specify the validation middleware for the secure upgrade of information. Next, we describe our case study in Section V and the evaluation in the vehicle simulator in Section VI. Finally, we draw a conclusion in Section VII.

II. Basic Concepts and Related Work

A. Replica Determinism and Voting

In a *deterministic* computer system, it is possible to predict a future state given its initial state and all timed future inputs. We call a system *replica deterministic*, if *it can be guaranteed by design that all correctly operating replicated components will visit the same states at about the same time, i.e., they will take the same decisions at all major decision points* [1]. Two replica–deterministic components (i.e., replicas) will produce the same service (i.e., intended behavior [2]) within some defined temporal bounds. A difference in the behavior of two replicas can be used an input for the fault detection in an *active redundancy* scheme. Basically,

Manuscript received June 21, 2012; revised July 22, 2012.

The authors are with the Institute for Computer Engineering, Vienna University of Technology, Austria (e-mail: armin.wasicek@tuwien.ac.at).

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

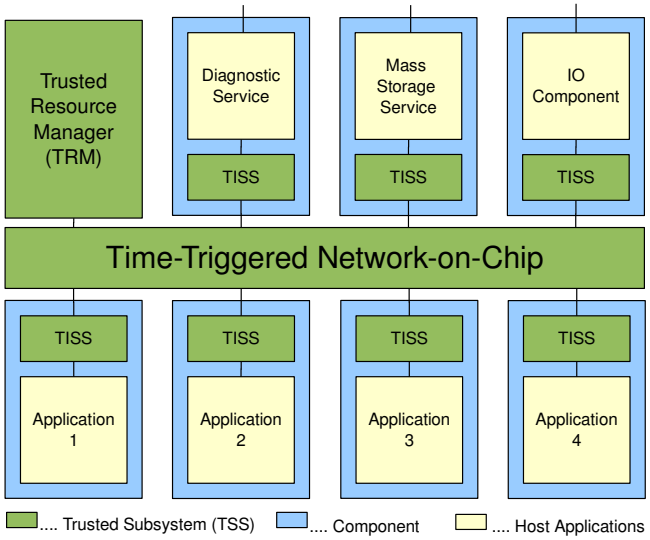

Fig. 1. Basic Layout of the ACROSS MPSoC

we distinguish between two classes of voting strategies for error detection between replicas:

- *Exact Voting* performs a bit–wise comparison of the bit patterns of its inputs
- *Inexact Voting* techniques introduce thresholds to determine approximate equivalence of results

B. The ACROSS MPSoC Architecture

The ACROSS architecture is the realization of a MPSoC architecture for heterogeneous application cores that provides a deterministic on–chip communication. It targets applications across different domains (e.g., avionics, automotive, industrial). Its particular benefits for application designers encompass fault isolation at the level of application cores, system–wide global time base to coordinate activities in the distributed system, and a temporal firewall interface to decouple computation from communication.

1) Basic Architectural Elements: The basic architectural elements of an ACROSS MPSoC [3] are depicted in Figure 1. *Components* are connected through a Time-triggered Network–on–a–Chip (TTNoC). A component is considered to be a self-contained computational element with its own hardware (processor, memory, communication interface, and interface to the physical environment) and software (application programs, operating system), which interacts with other components by exchanging messages. The *TTNoC* transmits messages according to an a–priori defined time–triggered schedule. The *Trusted Interface Subsystems (TISSs)* are the interaction points between the TTNoC and a component. The endpoints inside the TISS are called *ports*. Every TISS can have multiple input and output ports, but can not send or receive simultaneously. The *Trusted Resource Manager (TRM)* is a dedicated system component that manages the ports and routes on the TTNoC.

A *job* is a constituting element of a Distributed Application Subsystem (DAS) and forms the basic unit of computation. It interacts with other jobs through the exchange of messages. A DAS is a nearly independent distributed subsystem of a large distributed real-time system that provides a well–specified application service (e.g., a power-train system or a multimedia system) [4]. Jobs are allocated to components.

2) Security through Encapsulation: An *encapsulated communication channel* is an unidirectional data channel which transports messages at pre–defined points in time. The encapsulation mechanisms prevent temporal and spatial interference between different components. For instance, delaying or overwriting a message is not possible.

In an ACROSS MPSoC, encapsulated communication channels are established between TISSs through the time–triggered message transfer of the TTNoC. TTNoC accesses are arbitrated between different components through a Time–Division Multiple Access (TDMA) scheme. Every component implements its own local memory. Because no component can neither directly interfere with communication (e.g., by deliberately sending messages at certain instants), nor change a channel's configuration (these are exclusively managed by the TRM), an ACROSS MPSoC implements *segregation in the temporal and spatial domain*.

The encapsulation mechanism of the ACROSS MPSoC are not only beneficial to enforce its dependability properties, but also to efficiently implement security mechanisms. The paper in [5] discusses the capabilities of the ACROSS architecture to fulfill the basic requirements of a Multiple Independent Levels of Security (MILS) system [6]. The Trusted Subsystem (TSS) of the ACROSS architecture encompasses TTNoC, TISS, TRM and implements a *Separation Kernel* that isolates processes in separate partitions (components).

C. Totel's Integrity Model

Integrity models have been used for a long time in computer systems. Historically relevant integrity models are the Bell–LaPadula model for confidentiality [7] and the Biba model for integrity [8]. It is common to most integrity models to vertically subdivide a system into integrity levels I that are related with a partial order relation ($\leq$). The system designer assigns the tasks, applications, or subsystems (i.e., objects O) to a particular integrity level. Formally, the function $il : O \rightarrow I$ associates an integrity level to the system's components. The integrity model then defines how the components (e.g., subjects and objects) may interact. For instance, the Biba model defines that a subject should not be allowed to read an object of a higher integrity level (no write up), and to not read one of a lower level (no read down).

Totel's model [9] is a more recent development related to the Biba model. Contrary to Biba, where subjects access objects, Totel's model has only one kind of entities called objects. These objects provide *services* that can be requested by a client. Each object is classified within a particular integrity level that indicates to what degree it can be trusted and what its dependability requirements are. If an object creates a message, this message inherits its creator's integrity level. On the reception of a message, rules are applied to check if this message is valid or not.

The concept of a Validation Object (VO) is central to Totel's model. A VO *takes low level inputs and runs fault tolerance mechanisms to produce high integrity outputs* [9]. It reflects the circumstance that information flows that would be normally regarded as illegal, are needed for a flexible application design. The solution that Totel's integrity model advocates is to increase the trustability of the information contained in lower level objects to make the information useable at higher levels without corrupting higher level objects.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The fault-tolerance mechanisms to implement this upgrading are strongly application–dependent. In this paper, we present a case study to highlight one promising application of VOs in an automotive context.

The paper in [10] applies Totel's model in an avionics context. It serves as a model and a motivation for our work. Our work differs from [10] in following points:

- We consider input of sensor values rather than human input to a validation object
- The collected sensor data has real–time constraints
- Our communication system provides replica determinism and hence simplifies voting
- This work's context is automotive rather than avionics systems

III. Secure Information Sharing Subsystem

In this section we describe how to enable the secure sharing of information between different levels of criticality within an ACROSS MPSoC.

A. Definitions and Notations

We define our integrity model as a tuple $\langle J, C, M, I, il \rangle$:

- a set J of jobs
- a set C of components of an MPSoC
- a set M of messages on the TTNoC
- an ordered set I of integrity levels, with a partial order relation ($\leq$) between its elements
- $il : C \rightarrow I$, an association of a component to an integrity level

In addition, we will use these two functions:

- $val : M \rightarrow M \cup \{\epsilon\}$, a validation function
- $cr : M \rightarrow C$, an association between messages and their creating components

B. The ACROSS Integrity Model

The instantiation of an ACROSS MPSoC provides a basic platform to implement a multi–level secure system with different levels of criticality. The TSS of the ACROSS MPSoC acts as Trusted Computing Base (TCB) to encapsulate communication flows and to segregate criticality levels and components. The encapsulation within the architecture is achieved by strictly assigning one job to one component. Integrity levels are per definition assigned to components.

Rule 1: One job is assigned to exactly one component:
$\exists! j \in J, \exists! c \in C,\ j \rightarrow c$

The Unidirectionality property of an encapsulated communication channel is essential to implement Biba's rules:

Rule 2: Information flow is allowed only between components on the same or to a lower criticality level:
$\forall (c_1, c_2) \in C \times C,\quad c_1\ send\ c_2 \Rightarrow il(c_1) \geq il(c_2)$

Figure 2 depicts an example instantiation of the ACROSS integrity model. Each component on the MPSoC is assigned a desired integrity level. Next, each job is allocated to a component (rule 1). The arrows represent the encapsulated communication channels connecting the components (using rule 2). They all pass through the TSS. If a communication

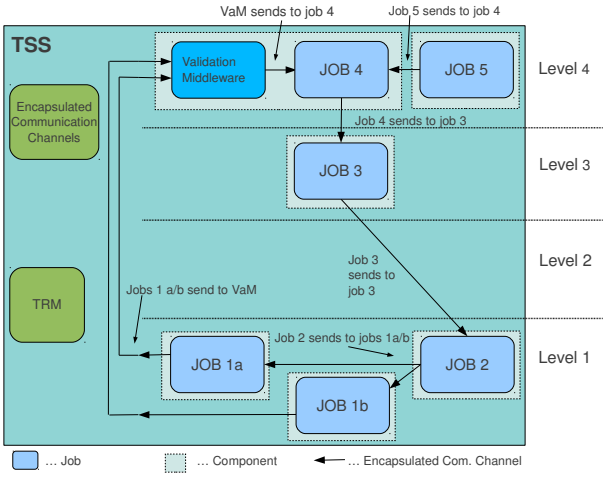

Fig. 2. The Integrity Model for ACROSS

path is required from a lower level component to a component at a higher level, a Validation Middleware (VaM) has to be placed within the receiving component. The VaM upgrades the information from several sources by applying a validation function.

Rule 3: Information flows from lower to higher integrity levels must pass through a VaM:
$C_{low} send c_{high} \Rightarrow$
(a) $il(c_i) < il(c_{high}), c_i \in C_{low} \subseteq C$
(b) $M_{in} = \{m_i \in em(C_{low})\}$
(c) $M_s \subseteq M_{in} | m_i, m_j \in M_s, |z(m_i) - z(m_j)| < \delta, m_i \neq m_j$
(d) $val(M_s) = M_v, \forall m_i \in M_v, m_i \neq \epsilon$

Note: Criterion (a) states that all incoming messages are received within a guaranteed upper bound, i.e., they are ordered within a specified time interval. Criteria (b) and (c) postulate that each message origins by a different, diverse or redundant sending component. Finally, criterion (d) requires that validation functions are only applied to transmissions from a lower level to a higher one.

C. Differences to Totel's Model

Basically, the definition of the ACROSS integrity model relies on the concepts of Totel's model. Because Totel's model is thought to be used in a single processor system, it has to be adapted for the use within an MPSoC architecture like ACROSS.

1) No integrity kernel required: The original model requires upward communication to pass through an integrity kernel. Why? In ACROSS, all communication is passing through the TSS which is considered to be a TCB. Moreover, because all information flows are clearly defined and rigorously checked at runtime by the TRM, there is no possibility that a covert channel can exist.

2) No Multi Level Objects (MLOs) defined: In the ACROSS integrity model, the concept of an MLO does not fit into the system design. The encapsulation of architecture enforces a strict assignment of one object to one component and one integrity level. The architecture by itself cannot guarantee that concurrent invocations of an MLO within a single component do not influence each other, because a component

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

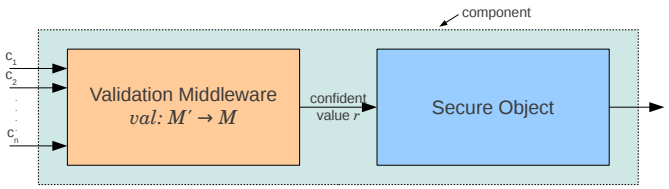

Fig. 3. Detailed block diagram of Validation Middleware

TABLE I
VALIDATION MIDDLEWARE ALGORITHM OVERVIEW

Complexity	Time	Space	Ref.
k^{th} Near. Neighbour w. Δ–Value	$O(n^2)$	$O(n)$	[13]
Probabilistic Boxplot Method	$O(n\,log(n))$	$O(n)$	[14]
Histogram Method	$O(n)$	$O(n)$	
Single-Linkage Clustering	$O(n^2)$	$O(n^2)$	[15]

is considered as an atomic unit. Accesses to the TISS ports from within a component are not arbitrated by architectural means. However, if additional segregation mechanisms are implemented within a component (e.g., through a partitioning operating system), concurrent access to the TISS can be mediated and MLOs can be supported.Therefore, the distinction from Single Level Object (SLO) and MLO is not necessary.

3) No integrity checks required: The invocation model of Totel assumes that object invocations are implemented through message–passing. A message is on the same level of integrity as its creator and carries a label indicating its level. In ACROSS these labels are not required, because messages can only be transmitted via encapsulated communication channels. The rules of the ACROSS require that channels are only established between objects on the same level. These rules have to be applied at design time and the strict adherence to these rules is enforced by the TRM.

4) No read–write rule: Each port in the TISS can be used either for reading or for writing, but it is not possible to perform both actions at the same time. Therefore, the read–write rule from Totel's model can be dropped in ACROSS.

IV. VALIDATION MIDDLWARE

This section explains how the previously proposed integrity model is used in combination with the ACROSS architecture.

Each component is assigned a level of criticality. Jobs are allocated to components and they produce messages that are then transmitted over encapsulated communication channels via the TSS. In case an upgrade of information between criticality levels is required (shown in Figure 2 between job1a/b and job 4), a VaM is inserted. The VaM gathers and processes information from redundant and diverse sources to produce the upgraded information.

A. Design of the Validation Middleware

Figure 3 depicts a VaM's basic information flow which provides N different and potentially diverse inputs, and a single output r. The primary purpose of a VaM is to evaluate the validation function val. In order to produce a meaningful output the input channels need to be completely independent from each other. An approach to achieve this independence is called N-Version Programming (NVP) [11], which relies on *diverse* (i.e., functionally equivalent programs that are independently generated from the same initial specifications) implementations.

The most common and intuitive way of comparing the multiple inputs of a VaM is a majority voting algorithm, but this creates some difficulties [12]. In some situations, the inputs need to be compared and determined to be correct, even if the input data differs. This raises the need for an inexact voter. Hence, inexact voting algorithms represent a solution for the implementation of a VaM. The requirements for these algorithms follow below.

B. Proposed Anomaly Detection Algorithms

The algorithms used to implement a VaM should be able to detect errors and prevent wrong values from propagating into the secure area of the application. The most important requirement of a VaM is that it has to be certified at the same integrity level as the application in the component itself. Therefore, it is desirable to keep this middleware simple and reusable. All algorithms should run online, possibly without knowing anything from the previous run. This kind of memory–less algorithms alleviate verification because the number of possible outcomes of a set of input data is always the same. The sources of data (i.e., sensors) often induce noise or even omit a value. For this purpose, they should provide filter mechanisms by preprocessing the input data. The prevalent resource constraints in embedded systems call for a light–weight approach with a low memory and computation footprint.

We propose a set of four anomaly detection algorithms that comply with these requirements. Table I summarizes their time and space complexity properties. We present an analysis of these algorithms in an automative environment in Section VI.

V. AUTOMOTIVE CASE STUDY

In this section we present an automotive case study as a proof–of–concept of the presented integrity model and to evaluate different algorithms implementing the case study.

A. Relevant Automotive Subsystems

We choose two related car functions with mixed–criticality requirements for safety and security for our case study:

1) Odometer Subsystem: An *odometer* computes and stores the current mileage counter of a vehicle. In order to log the distance covered by the car, the current speed needs to be sampled and multiplied with the time elapsed since the last measurement. The accumulated results of this computation is the total distance traveled. This subsystem has a high security requirement, because a vehicle's resale value depends largely on this value. Odometer fraud is the illegal practice of rolling back odometers to make it appear as if vehicles have lower mileage than they actually have [16].

2) ABS Subsystem: The second subsystem is an Anti–Blocking System (ABS). The ABS prevents wheel lock–up during heavy braking [17]. This system is introduced to increase safety and to prevent the abrasion of the wheels. In practice, the ABS controller detects the wheel lock-up as a

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

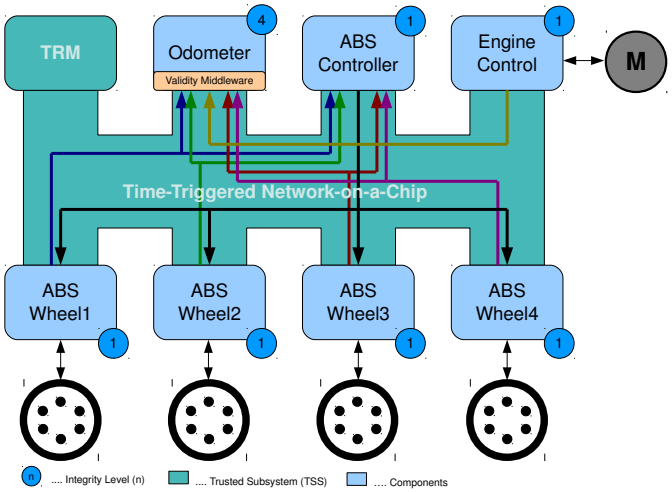

Fig. 4. Allocation of automotive subsystems to ACROSS MPSoCs

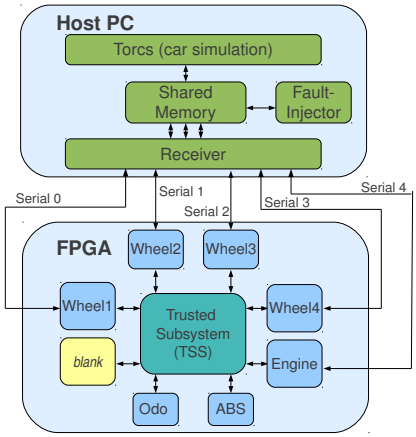

Fig. 5. Block diagram of simulation environment

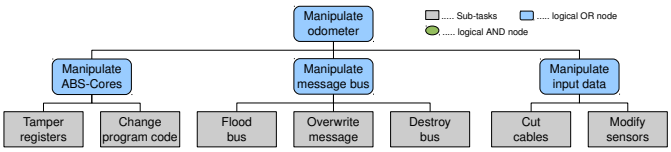

Fig. 6. Attack model for the odometer subsystem

sharp increase in wheel deceleration. In our case study, a four channel, four sensor system is used, which has a speed sensor on each wheel and separate valves to apply a brake force to each wheel. The ABS controller computes an appropriate brake force for each wheel, if this value is below a fixed ABS slip limit. This new brake force value replaces the value sensed from the brake pedal. Security requirements are not common in ABSs.

3) Integration of subsystems: The traditional setup of a car is to deploy the ABS and odometer subsystems in a federated approach by physically disjoint systems. The advantage of such an approach is that the segregation between subsystems is high. However, this comes at a high cost of many redundant units. Integrated architectures like ACROSS aim to deliver similar segregation properties present in a federated system and simultaneously reduce size, weight, power consumption of the hardware while reducing complexity and increasing re-usability of the software [18]. The benefits of integrated architectures are leveraged through resource sharing. In this case study, we implement the ABS and the odometer subsystems in a single ACROSS MPSoC. Figure 4 depicts the mapping of the automotive subsystems onto the chip. We speak of the resulting system as a mixed-criticality system, because both subsystems have different safety and security requirements. Besides the benefit of using the same hardware infrastructure, both subsystems can share the measured speed values. We design our system such that the odometer subsystem uses the speed values of the ABS and engine speed sensors.

B. Simulation Environment

We developed a hardware–in–the–loop system based on the The Open Racing Car Simulator (TORCS)[1] to evaluate our concepts (see Figure 5). TORCS provides a realistic physics environment, car models, race tracks, and an interface to program robots, which are programs that steer the vehicle.

The simulation environment is divided into two parts: The car and the environmental simulations run on a standard PC. The car simulation communicates with an Field–Programmable Gate Array (FPGA) that implements an ACROSS MPSoC and executes the car's ABS and odometer subsystems. To establish a relatively simple connection from the host PC to the FPGA, these two parts are connected through five autonomous serial interfaces. The data exchange between both parts is facilitated via a shared memory. A repeater program on the PC polls the serial devices, checks the data integrity and stores the data into the shared memory. We used a Linux–based Dell Optiplex 755 for a PC and a Altera Stratix III^{TM} Development Kit for the FPGA.

The $Wheel[1..4]$ jobs implement speed sensors and brake actuators. The $Engine$ job implements the speed sensor of the engine controller. They receive their sample values from the environmental simulation via the serial line and forward it to the ABS controller job and the odometer job via the TSS according to the time–triggered schedule. The ABS controller computes the actual brake force value and forwards that value to the $Wheel[1..4]$ jobs which again apply this brake force by transmitting a brake force value to the car simulator. The odometer job computes and stores the current mileage.

The update frequency of the car simulation is 22 ms. The periodicity of the messages on the TTNoC is set to 15 milliseconds. This provides enough accuracy for the car simulation run smoothly.

C. Odometer Attack Model

We use the attack tree method [19] to assess potential attacks on the odometer task (see Figure 6). There are three main parts of the odometer subsystem that are viable attack targets:

1) *Manipulate jobs:* This can be achieved either by overwriting the registers that hold the speed values of the sensors, or by manipulating the program running on the ABS components of the wheels (e.g., downloading a modified version). This branch of the attack tree can be prevented by software security mechanisms which are out of scope of this paper.

[1]http://torcs.sourceforge.net/

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

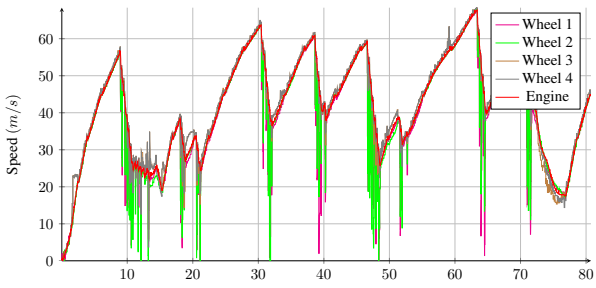

Fig. 7. All five speed values during one lap of a race

2) *Manipulate communication system:* This can be carried out through a flooding attack to block the communication or to periodically overwrite messages to induce wrong speed information. These attacks are prevented by the encapsulation mechanisms of the architecture.
3) *Manipulate input data:* Five redundant and diverse sensors are used to provide the current speed of the vehicle. The simplest manipulation is to cut the cables which connect the sensors with the communication system (i.e., create a stuck–at–value). Another way is to tamper with the measurement of the installed sensor. If the odometer depends on a single sensor (as it is the state–of–the–art), it is attackable. If the odometer depends on five redundant and diverse sensors like in our case study, its manipulation is hindered. If in addition, these sensors are used in conjunction with the safety–critical part of the system, an attacker has might be refrained from tampering the sensor devices of the ABS subsystem.

D. Software–based fault injection

The simulation environment supports software–based fault injection by modifying the contents of the shared memory. The following faults can be simulated:

- The wheel sensor can be disabled, which simply freezes the value at the last measured value.
- The current speed of a sensor can be set to a fixed level.
- The ABS subsystem can be enabled or disabled.
- A fault injection schedule containing the name of the faulty sensor, the start- and end times and the modified speed value can be defined. This schedule is executed during runtime.

VI. Evaluation

We use the simulation environment presented in the previous section to evaluate the proposed secure information sharing system. We assume the rightmost branch ('Modify Sensors') of the attack tree, hence, an attacker is tampering with the physical interface of a sensor. Figure 7 depicts the evolution of a vehicle's speed during a sample race.

The deviation of the speed sensor values is the main testing criteria for the anomaly detection algorithms used in the VaM. The algorithms need to find a majority of speed values with a small variance and eliminate possibly wrong or inaccurate values. The optimal behavior of an algorithm is that it always finds a manipulated speed value, which would increase or decrease the current value of a sensor and therefore change the current mileage. Secondly it should neglect values from wheels which are currently locked up or spinning, because these values would also influence the computed mileage.

TABLE II
Percentage of results determined as valid

	no faults injected	faults injected
k^{th} Nearest Neighbor	93.58 %	88.49 %
Boxplot	99.96 %	99.98 %
Histogram	96.04 %	89.19 %
Singe-Linkage Clustering	100.00 %	100.00 %

We conducted several experiments (with and without fault injection) to study the proposed algorithms. Table II gives a brief summary of the outcome of the experiment runs with and without fault injection.

Failure or manipulation of wheel sensors is tolerable as long as a majority of sensors is still correct. If a majority of values is incorrect and not equal, the behavior of each algorithm is different. The k^{th} nearest neighbor method, the probabilistic boxplot method and the histogram method find no valid result and therefore reuse the last valid value. The single-linkage clustering algorithm merges the values until a majority is found, even if other algorithms would have marked some values as wrong. Because of this, the clustering algorithm always produces an output. The optimal solution for dealing with a missing majority is different and depends on the application. The duration during which no majority is found can be very long and therefore the last valid value would have to be reused for a long period. This could cause a huge deviation and it can produce a wrong result.

If an algorithm always finds a solution it maybe has to use values where the deviation from the real speed value is so big that it influences the final result. Because the average of all values in the majority set is used and since there are some correct values in, the deviation of the end result is reduced by the correct values in the majority.

The worst case of the whole fault scenario occurs when wrong values happen to be the same and they form a majority. Then the valid values are a minority and the faulty values alone determine the result.

Looking at the exemplary automotive application with the five speed sensors, two faulty sensors can be detected. The main problem in this application is the breaking scenario, where a huge deviation between the different speed values can arise. This can occur when the wheels are locked-up or spinning during a braking maneuver. If there are two faults injected concurrently, both with a low brake force value, and if the wheels are blocking and produce a low value at the same time, the speed of the car can get nearly zero even if it is still driving with a high speed. This kind of scenario points out a possible problem which exists for all four analyzed algorithms. A solution would be to use another class of anomaly detection algorithms. For instance, algorithms which are able to learn the optimal behavior for every situation. As most of these algorithms require working memory, they are difficult to certify and are therefore difficult to deploy in safety–critical environments.

VII. Conclusion

In a mixed–criticality system with multiple security levels the induction of faults in a secure component is detected and prevented through the deployment of a VaM. The VaM is

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

used to upgrade the integrity of this information flow. Therefore, the VaM needs diverse and redundant inputs to upgrade the information. We propose to use inexact voting based on anomaly detection algorithms which do not aggregate values between subsequent executions. This is a particular requirement to facilitate certification. However, in extreme driving scenarios (e.g., braking), when all sensors show different readings, these algorithms do not work accurately.

Most important, to realize the proposed secure information sharing system, is the availability of diverse and redundant inputs that are then used to upgrade information. A particular strong point of our design is that we are able to separate the execution platform configuration (including the configuration of the integrity model) and the computation: platform configuration can be certified independently from jobs. Summarizing, our integrity model to securely exchange information between different criticality levels can be efficiently implemented in an integrated architecture like ACROSS. Certifiable anomaly detection algorithms which actually perform the upgrade require further research.

ACKNOWLEDGMENT

This document is based on the ACROSS project in the framework of the ARTEMIS program. The work has been funded in part by the ARTEMIS Joint Undertaking and National Funding Agencies of Austria, Germany, Italy and France under the funding ID ARTEMIS-2009-1-100208. The responsibility for the content rests with the authors. The authors would like to thank Jean Arlat and Youssef Laarouchi for the insightful discussions.

REFERENCES

[1] H. Kopetz, "Why time-triggered architectures will succeed in large hard real-time systems," in *FTDCS*, 1995, pp. 2–9.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, January 2004.

[3] C. El-Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek, "The ACROSS MPSoC – A New Generation of Multi-Core Processors designed for Safety-Critical Embedded Systems," in *Proceedings of the 15th Euromicro Conference on Digital Systems Design (DSD)*, 2012.

[4] R. Obermaisser and H. Kopetz, Eds., *GENESYS: A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Süd dwestdeutscher Verlag für Hochschulschriften (SVH), 2009.

[5] A. Wasicek and C. E. Salloum, "A system–on–a–chip platform for mixed–criticality applications," in *Proceedings of 13th IEEE International Symposium on Object/component/service-oriented Real-time distributed computing (ISORC)*, May. 2010.

[6] C. Boettcher, R. DeLong, J. Rushby, and W. Sifre, "The MILS Component Integration Approach to Secure Information Sharing," in *Proceedings of the 27th Digital Avionics Systems Conference (DASC)*. IEEE/AIAA, October 2008.

[7] D. E. Bell and L. J. LaPadula, "Computer Security Model: Unified Exposition And Multics Interpretation," MITRE Corp., Bedford, Tech. Rep., June 1975.

[8] K. J. Biba, "Integrity Considerations For Secure Computer Systems," Mitre Corporation, Tech. Rep., April 1977.

[9] E. Totel, J.-P. Blanquart, Y. Deswarte, and D. Powell, "Supporting Multiple Levels of Criticality," *ESPRIT project 20716: GUARDS*, 2000.

[10] Y. Laarouchi, Y. Deswarte, D. Powell, and J. Arlat, "Connecting Commercial Computers to Avionics Systems," *28th Digital Avionics Systems Conference*, pp. 6.D.1–(1–9), Dec. 2009.

[11] A. A. Avizienis, "The Methodology of N-Version Programming," *Software Fault Tolerance edited by M. Lyu, John Wiley & Sons*, pp. 23–46, 1995.

[12] P. R. Lorczak, A. K. Caglayan, and D. E. Eckhardt, "A Theoretical Investigation of Generalized Voters for Redundant Systems," *Digest of Papers FTCS-19:The Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 444–450, 1989.

[13] E. M. Knorr, R. T. Ng, and Tukakov, "Distance-based Outliers: Algorithms and Applications," *The VLDB Journal 8*, pp. 237–253, 2000.

[14] R. Mcgill, J. W. Tukey, and W. A. Larsen, "Variations of Box Plots," *The American Statistician*, vol. 32, pp. 12–16, 1978.

[15] M. Matteucci, "Hierarchical clustering algorithms," 2000, available at: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html.

[16] "Preliminary report: The incidence rate of odometer fraud," National Highway Traffic Safety Administration (NHTSA), Tech. Rep. DOT HS 809 441, 2002.

[17] D. Burton, A. Delaney, S. Newstead, D. Logan, and B. Fields, "Effectiveness of ABS and Vehicle Stability Control Systems," Royal Automobile Club of Victoria (RACV) Ltd, Tech. Rep., April 2004.

[18] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 956 –965, july 2009.

[19] T. R. Ingoldsby and C. McLellan, "Creating secure systems through attack tree modeling," Amenaza Technologies Limited, 550, 1000, 8^{th} Ave SW, Calgary, AB, Canada, Tech. Rep., June 2003.