Application of Software Watchdog as a Dependability Software Service for Automotive Safety Relevant Systems

Xi Chen
DaimlerChrysler AG – Stuttgart, Germany
xi.chen@daimlerchrysler.com

Juejing Feng
RWTH Aachen University – Aachen, Germany
juejing.feng@rwth-aachen.de

Martin Hiller
Volvo Technology Corporation – Gothenburg, Sweden
martin.hiller@volvo.com

Vera Lauer
DaimlerChrysler AG – Stuttgart, Germany
vera.lauer@daimlerchrysler.com

Abstract

To face the challenges resulting from the increasing density of application software components and higher dependability requirements of the future safety systems in the automotive electronics, a dependability software service to monitor individual application software components in runtime is required in order to improve the overall system dependability. This paper proposes the application of a Software Watchdog service providing heartbeat monitoring and program flow checking. The Software Watchdog is integrated in a software platform for the automotive safety electronics. A model-based design with Matlab/Simulink and an evaluation of this Software Watchdog service in a hardware-in-the-loop validator are also given.

Index Terms — Software Watchdog, heartbeat monitoring, program flow checking, automotive dependable software platform and software services

1. Introduction and background

The on-board electrics and electronics (E/E) in vehicles have become subject to more demanding requirements in recent years, triggered by an ever increasing demand for safety and comfort. On the one hand, the complexity and quantity of applications implemented by electronics and software is increasing dramatically [1]. On the other hand, E/E systems and networks should exhibit at least the same dependability as state-of-the-art mechanical systems with comparable cost.

With the introduction of powerful microcontrollers and for reasons of cost, the number of ECUs (Electronic Control Unit) will be consolidated and even reduced as more and more functions will be integrated on one ECU. Facing this challenge, defining a common software platform and standardization of software services are two key approaches.

One central standardization initiative is the AUTOSAR Consortium [2], which aims to provide a standardized software platform for each in-vehicle ECU. According to the AUTOSAR vision, future embedded software will be developed independently from ECU hardware details. Application software components can be mapped onto different ECUs, where the application can be divided into code sequence components called runnables. Runnables from different applications can be mapped onto the same task, while tasks from different applications can also be mapped onto the same ECU. Therefore, it is possible, that runnables from different software components can be mapped to the same task.

Considering the dependability requirements of the runnables, different time constraints can be required for two runnables. Those runnables should be treated differently in fault detection and error processing.

Taking the current trends into consideration, we need more powerful services to detect timing and program flow faults. Such services are not only required in the system integration and validation phases but also in runtime. A well-defined layered software platform [3] with flexible dependability software services for fault tolerance is a key issue for providing safety-relevant control systems in automotive applications.

The Software Watchdog service, aimed for monitoring the timing behavior and program flow, is designed in the scope of the EU project EASIS (Electronic Architecture and System Engineering for Integrated Safety Systems, see www.easis.org). This project is an industry consortium of 22 European leading vehicle manufacturers, system suppliers, tool producers and research institutes. Today, safety systems are mainly stand-alone systems which have little or no links across domain borders (domains in this context are e.g. powertrain, chassis, and body). So-called Integrated Safety Systems (ISS) combine active and passive safety systems across domain borders. Links to telematics services are also included in the term.

2. Related work

For automotive safety systems, satisfying real-time requirements in a deterministic manner is a critical issue. To meet timing constraints, different monitoring mechanisms have been developed, such as the ECU hardware watchdog [3][5][6], deadline monitoring [7][8] and execution time monitoring [9]. Approaches of control flow checking techniques with signatures...
have been developed in the IT-industry to ensure the correct execution sequence of the programs.

A hardware watchdog treats the embedded software as a whole. With the increasing density of applications on one ECU, the hardware watchdog should be supplemented with software services for the monitoring execution on a more detailed level, i.e., on the level of tasks or runnables.

Deadline monitoring of the OSEKTime [8] operating system and execution time monitoring [9] of AUTOSAR OS introduce the time monitoring of tasks, but the granularity of fault detection on the layer of tasks is not fine enough for runnables.

In the field of control flow checking, a lot of work has been made in the IT-industry to check the correctness of program flow. Most of the current methods of control flow checking are based on assigning signatures to blocks as introduced in [10]. Such a technique suffers from high performance overhead and low flexibility with regard to modification of programs.

3. Functional design

The Software Watchdog is supposed to ensure the real-time characteristics of the system. Tracing the cause of the violation of real-time requirements and an early detection of timing faults are not easy because of the complicated situations leading to the timing errors. A violation of a timing constraint can be placed into one of the following two categories:

1. An object hangs as a result of a requested resource being blocked, either by the object itself or some other object, long enough to violate the timing constraint.
2. An object is excessively dispatched for execution.

According to the categorization, the Software Watchdog identifies those two situations by monitoring the aliveness and arrival rate of the runnables. For correct execution of tasks, program flow is checked by the Software Watchdog treating runnables as basic blocks.

Before the design concept of the Software Watchdog is introduced in detail, we will briefly describe the EASIS software platform, into which the Software Watchdog is integrated.

### 3.1. EASIS dependable software platform

The EASIS software platform [11] focuses on ECUs for the ISS-applications which cross domain borders. It separates the safety relevant application software and the underlying ECU hardware by providing a software platform with standard interfaces.

As shown in Figure 1, the activities concerning the EASIS software platform cover layers L2 through L4. The ISS dependability software services in L3 aim to enhance the safety, reliability, availability and security of new safety functions. ISS gateway services in L3 provide secured inter-domain communication services. An OSEK-conforming operating system [7] with safety relevant services such as the Software Watchdog is integrated across L2 and L3. Device drivers, microcontroller abstraction layer and fault tolerant hardware platform reside on L2 and L1, respectively.

![Figure 1: EASIS software topology](image)

### 3.2. Design of the Software Watchdog

The design of the Software Watchdog (as depicted in Figure 2) follows the concept of heartbeat monitoring of runnables. With the information provided by the heartbeats of the runnables, it is possible to monitor periodicity and execution sequence of the runnables. Furthermore, task state, application state and global ECU state can be derived. Thus, the Software Watchdog can be designed to inform other dependability software services in the EASIS software topology, such as the Fault Management Framework [12], a general fault handling service in the platform, to undertake different fault treatments depending on the source, type and severity of the detected faults.
The Software Watchdog has three basic units: the heartbeat monitoring unit, the program flow checking unit and the task state indication unit.

- **Heartbeat monitoring unit**: With the help of a heartbeat indication routine, runnables report their heartbeats to the Software Watchdog.
- **Program flow checking (PFC) unit**: The program flow checking service monitors the execution sequence of runnables by comparing real executed successors with a predefined set of possible successors of the predecessors.
- **Task state indication (TSI) unit**: Errors of monitored runnables detected in the upper two blocks will be reported to the TSI unit. The TSI unit then compares the number of detected errors with some pre-defined thresholds and generates individual supervision reports on runnables. These reports can be used to derive error indication states of the tasks, which in turn can be used for determining the status of the applications.

### 3.2.1. Heartbeat monitoring

Heartbeat monitoring offers a mechanism for periodically monitoring the aliveness and arrival rate of independent runnables. The following fault types are handled:

- **Fault type for aliveness monitoring**: The runnable is blocked or preempted for some reason and its aliveness indication routine is not executed frequently enough.
- **Fault type for arrival rate monitoring**: Within one period, there are more aliveness indications of the monitored runnable than expected.

In EASIS, we chose a passive approach to record and monitor the runnable updates by saving the heartbeats of runnables in the Aliveness Counter (AC) and Arrival Rate Counter (ARC). These two kinds of counters are assigned to each runnable to record its heartbeats during the defined monitoring period according to the fault hypothesis and checked shortly before the next period begins. The monitoring periods are recorded in the Cycle Counter for Aliveness (CCA) and Cycle Counter for Arrival Rate (CCAR). All of those counters are reset to zero, if the periods defined in the fault hypothesis expire or an error is detected in the last cycle. In addition to the data resources used for the computation, an Activation Status (AS) is assigned to each runnable to control the heartbeat monitoring of the runnables.

### 3.2.2. Program flow checking

Correct program flow is a fundamental part of the correct execution of computer programs. In the embedded system, the following faults could cause program flow errors:

- Faults in the software design, implementation and/or integration.
- Transient faults in the system, e.g. memory errors.
- Corruption of the program counter.

To reduce the overhead involved during program flow checks as well as the system complexity, only the sequence of the safety-critical runnables will be monitored. With the help of aliveness indication routines, which are integrated into the runnables as automatically generated glue code, a view of which runnables are currently being executed can be derived. Compared with the widely discussed method of using embedded signatures as proposed in [10], or with the watchdog processor as discussed in [13], a simple approach with a look-up table was applied to minimize performance penalty and extensive modification requirements of applications. In the look-up table, all possible predecessor/successor relationships of the monitored runnables are stored and compared to the actual execution sequence.

### 3.2.3. Task state indication

In order to achieve global monitoring by integration of the results from the monitoring of runnables, the error messages of runnables are recorded by the Task State Indication Unit in an error indication vector. If one of the elements in the error indication vector reaches the threshold, the whole task will be considered faulty. Based on the mapping information of applications and tasks, corresponding fault treatments with a global view of the ECU are taken:
If the global ECU state is “faulty”, the ECU might be subjected to a software reset depending on the requirements and constraints of applications.

If the global ECU state is “OK”, the “faulty” application software components might be restarted or terminated.

If there are other tasks, which do not belong to any of the terminated/restarted applications, these tasks might be terminated and restarted with the services provided by the operating system.

4. Validation

Before we explore the concept validation of the Software Watchdog in detail, a brief introduction to the EASIS validation activities, development process, methodology and prototyping tool chain will be given.

4.1. Brief introduction to EASIS validator

The EASIS architecture validator [14][15] focuses on the prototyping and validation of some of the most relevant properties of the EASIS architecture, including fault-tolerant hardware, dependability software architecture and services. The EASIS HIL (Hardware-in-the-Loop) validator hosts a number of ISS-applications, such as the driver assistance applications SafeLane and SafeSpeed with Steer-by-Wire technology. SafeLane is a lane departure warning application, and SafeSpeed is a system to automatically limit the vehicle speed to an externally commanded maximum value. The nodes in the architecture validator include fault-tolerant actuator and sensor nodes, driving dynamics control, environment simulation, light control node and a gateway node, which connects different vehicle domains of TCP/IP, CAN (Controller Area Network) [16] and FlexRay [16].

4.2. Brief introduction to validation process

The validation of the Software Watchdog is performed on the central node of the EASIS architecture validator. The central node is an AutoBox, a rapid prototyping platform from dSPACE [17], where the safety applications of SafeSpeed with control algorithms and dependability software services e.g. the Software Watchdog are integrated.

As shown in Figure 3, the validation process of the Software Watchdog follows the model- and simulation based development process, in which a sub-system is developed and tested in a pure software environment with the help of simulation tools (Software-in-the-Loop test). The implementation of software on a particular hardware platform will only be initiated after a successful test of the functionalities with the help of simulation models. In the first step after requirements analysis of the applications, system functional design is initiated by building and modeling the whole system with Matlab/Simulink, in particular the modeling of runnables. The Software Watchdog is prototyped and simulated on a PC as a virtual prototype in the third step. In the next step, based on the knowledge gained from the virtual prototype and other automotive constraints such as memory and timing requirements, the AutoBox was chosen for the validation and evaluation of the concepts in the EASIS architecture validator. The virtual prototype of the Software Watchdog and the modeled safety application will be mapped onto tasks and scheduled on the system architecture. Individual hardware specific C-codes were generated, compiled and loaded onto the rapid prototyping platform.

4.3. Modeling, simulation and prototyping of the Software Watchdog

For the modeling of task dispatching and program flow of runnables in OSEK, Stateflow in Matlab/Simulink was applied. Stateflow is a design and development tool used for modeling complex system behavior based on finite state machines. Runnables are modeled with function-call subsystems and triggered by events sent by Stateflow in a defined execution sequence. A function-call subsystem is a block in Matlab/Simulink which can be invoked as a function by another block. For instance, as illustrated in Figure 4, the application SafeSpeed can be divided into three runnables: sensor value reading in GetSensorValue, the control algorithm in SAFE_CC_process and setting of the actuator in

Figure 3: Tool chain and development process
Speed process. These are triggered as function-call subsystems by the Stateflow chart SafeSpeed, in which the execution sequence of runnables is implemented. To indicate the aliveness of the runnables, further function-call subsystems to simulate the glue code are also implemented, which report the execution of the runnables.

The time-triggered behavior of the heartbeat monitoring unit and task state indication unit was modeled with time counters. Thus, in order to simulate the mechanism of task scheduling with different periods in the operating system, different time counters can be assigned to the Stateflow charts. On the other hand, the program flow checking unit was modeled using an event-triggered Stateflow chart.

4.4. Integration of the Software Watchdog in the EASIS software platform

Following the layered architecture of the EASIS software platform (see Figure 1), the Software Watchdog service is integrated into L3 as a separate module with defined interfaces to other software modules. There are two main interfaces to the Software Watchdog. The first interface serves for application software components in L1 to report their aliveness indications to the Software Watchdog. The other interface is used for the Software Watchdog to report the detected faults to the Fault Management Framework. Fault Management Framework is a general fault treatment system that gathers the information on the detected faults, and informs the applications about the fault detection. Lastly, coordinated fault treatment can be carried out with the help of the Operating System and Fault Management Framework.

4.5. Evaluation of the Software Watchdog in EASIS validator

The evaluation of the Software Watchdog is performed based on the fault/error definition in the design phase. Since different faults can result in the same error, error injection is applied for the evaluation of the design and prototyping of the Software Watchdog. Such an approach has the advantage that the dependability requirements can be tested in a front-loading manner of system development. The concept can be validated independently from the specific fault-types. Faults, which are difficult to inject into the test bench or on-road test, can be relatively easily emulated with errors.

Here again Stateflow is used to manipulate the execution frequency and sequence of runnables by changing the timing parameter of runnables, manipulation of loop counters and building invalid execution branches, etc. The experiment environment ControlDesk [17] from dSPACE provides the possibility to manipulate the data assigned to the timing parameter of runnables to the condition that determine the invalid execution branches in the runtime. Therefore, it is used to trigger the error injection during the execution of the applications and visualize the results as well.

By building different evaluation cases, the three chief functionalities of the Software Watchdog, i.e. the detection of the aliveness error, the arrival rate error and the program flow error, are successfully validated. The following screenshots demonstrate some of the evaluation cases generated by injecting heartbeat or program flow errors. The x-axes of each plot in the diagram indicate the time lapse, which has a scalar of 10ms. The y-axes indicate the value of the counter and number of detected error.

In order to inject heartbeat errors, a time scalar is connected to a slider instrument to change the execution frequency. For example, Figure 5 shows the test with an injected aliveness error. Similar test with arrival rate error and control flow error were performed as well. The increase in the y-value in the last plot “AM Result” (Aliveness Monitoring Result) indicates the detection of the errors.

Figure 5: Test with injected aliveness error

Figure 6 shows the case in which the real cause of the erroneous state is identified through the collaboration of the units of the Software Watchdog. Here, the aliveness errors detected by the heartbeat
monitoring unit are actually caused by program flow errors, which are reported with the plot “PFC Result” (Program Flow Checking Result). After the detection of three program flow errors (which here is set as the threshold), the task state is set to “faulty”. Only one accumulated aliveness error is reported.

Figure 6: Collaboration of fault detection units

5. Conclusions and outlook

Conclusions
The concepts and design of the Software Watchdog proposed in this paper reflects the current trends in automotive software development.

The Software Watchdog, a software-implemented dependability service, monitors the individual timing constraints of application runnables and their program flow. It demonstrates the functional potential for improving dependability in distributed in-vehicle embedded systems. The interface of the Software Watchdog provides information for further fault treatments and variants for fault containment and tolerance.

Many valuable experiences were gained during the modeling, simulation and rapid prototyping of ISS applications and dependability software services using the concept of runnables.

Outlook
In the EASIS architecture validator, further analysis of fault detection coverage can be of interest for the mapping and application of the Software Watchdog to meet the individual dependability requirements of different safety systems. The functionalities and performance of the Software Watchdog with regard to fault handling strategies, especially concerning dynamic reconfiguration of applications and collaboration of monitoring units are further evaluated on an evaluation microcontroller S12XF from Freescale.

References