2011-01-0448

An AUTOSAR-Compliant Automotive Platform for Meeting Reliability and Timing Constraints

Author, co-author list (Please ensure the Participant tab in MyTechZone matches the author information entered here including author/co-author order. SAE will populate the information in this section from MyTechZone when publishing your final paper)

Affiliation (Please ensure the Participant tab in MyTechZone matches the affiliation information entered here including author/co-author order. SAE will populate the information in this section from MyTechZone when publishing your final paper)

Author, co-author list (Please ensure the Participant tab in MyTechZone matches the author information entered here including author/co-author order. SAE will populate the information in this section from MyTechZone when publishing your final paper)

Affiliation (Please ensure the Participant tab in MyTechZone matches the affiliation information entered here including author/co-author order. SAE will populate the information in this section from MyTechZone when publishing your final paper)

ABSTRACT

High demands on advanced safety and driving functions, such as active safety and lane departure warnings, increase a vehicle's dependency on automotive electrical/electronic architectures. Hard real-time requirements and high reliability constraints must be satisfied for the correct functioning of these safety-critical features, which can be achieved by using the AUTOSAR (Automotive Open System Architecture) standard. The AUTOSAR standard was introduced to simplify automotive system design while offering inter-operability, scalability, extensibility, and flexibility. The current version of AUTOSAR does not assist in the replication of tasks for recovering from task failures. Instead, the standard assumes that architecture designers will introduce custom extensions to meet such reliability needs. The introduction of affordable techniques with predictable properties for meeting reliability requirements will prove to be very valuable in future versions of AUTOSAR.

In this paper, we propose a new Software-Component (SW-C) allocation algorithm called R-FLOW (Reliable application-FLOW-aware SW-C partitioning algorithm) for fail-stop processors to support fault-tolerance with bounded recovery times, and we integrate the R-FLOW algorithm into AUTOSAR. R-FLOW leverages different types of replication schemes to satisfy reliability and timing constraints, while offering a high degree of resource utilization and flexibility. Specifically, R-FLOW classifies real-time periodic tasks into *Hard Recovery* tasks, *Soft Recovery* tasks, and *Best-Effort Recovery* tasks. *Hot Standbys* are used for recovering from failures of hard recovery tasks, whereas *Cold Standbys* are utilized for recovering from failures of soft recovery and best-effort recovery tasks. With this goal in mind, we design and implement our proposed architecture within the guidelines of the current AUTOSAR framework. We have built an at-scale prototyping platform, comprising of Freescale HCS12X processing boards, a dual-channel FlexRay bus, and a CAN network. Our proposed architecture is evaluated on this platform using reliability and timeliness metrics in the context of different fault scenarios.

INTRODUCTION

Continuing improvements on embedded systems encourage x-by-wire technology as well as various types of safety and comfort features in future vehicles [1]. In particular, safety features such as lane keeping, lane changing, collision avoidance and driver warning require high dependability because of their safety-critical nature. However, these features require complex hardware and software platforms, and the design and implementation of these features is a challenge. Moreover, because safety features are composed of several subsystems including sensors, processors, and actuators, the whole system needs to be carefully designed to avoid situations where, for example, a single defective sensor can cause an unintended event [12]. Modern multi-core processors can execute several applications in parallel and still the overall system needs to be designed such that single chip failures cannot result in an undesirable situation. Hence, system-level dependability is a key concern in rapidly evolving automotive industries.

Dependable systems can be implemented by using fault-tolerant techniques, and the conventional fault-tolerance technique is to replicate processes, either concurrently or sequentially. Graceful degradation can also contribute to system dependability [11]. However, graceful degradation can involve considering all possible failure scenarios, which can be an exponentially hard problem. Replication, such as Triple Modular Redundancy (TMR) for hardware and N-version programming for software, is a typical approach, but it requires more resources than graceful degradation. Although these techniques have been extensively used in domains such as avionics, space shuttles, and industrial facilities, they may not always be appropriate long-term solutions for automotive architectures due to these exorbitant costs. Our work presents resource-efficient techniques for achieving the required dependability.

Applications in the automotive system are closely connected to the physical environment and use sensor information to obtain current physical information. For example, in Steer-by-Wire (SBW) systems [3], sensors measure information about steering wheel movement, and computational components in microprocessors compute signals for controlling the wheels with the information from sensors. Actuators receive the control signals for the motors directly, and these signals are handled periodically for timely handling of user operations and reactions to the environment.

In order to reflect this nature, we define an *application flow*, which is composed of periodically executing runnables generating information data and events regularly that flow through multiple runnables. An application flow also has an end-to-end delay from input to output. Each runnable is represented by a periodic task [13], ρ_i, which releases a job every T_i units of time, where each job consumes at most C_i units of computation time and should be completed within a relative deadline, $D_i \leq T_i$.

Within an application flow, runnables are classified into sensor/actuator runnables and computational runnables. For instance, an actuator runnable controlling the steering wheel motors must run on the Electronic Control Unit (ECU) connected to the motors in an SBW system. Every runnable generates data to be fed to other runnables, except actuator runnables which terminate an application flow. Figure 1 and Figure 2 show an exemplary diagram of an application flow applied to the autonomous vehicle which won the DARPA Urban Challenge [17].

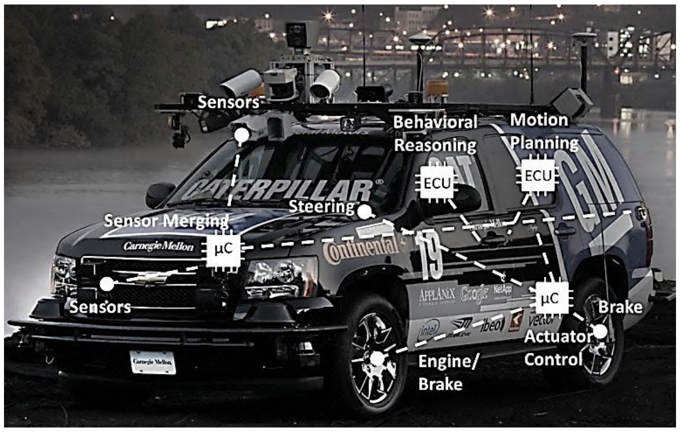

Figure 1: The application flow model can be applicable to a driverless car

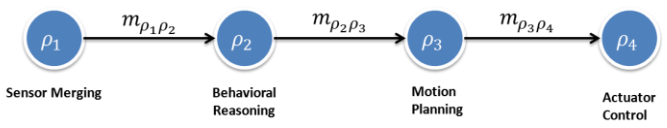

Figure 2: An exemplary Application flow from Figure 1, where ρ represents a runnable, and m denotes a message between two runnables

Handling failures on-demand with bounded recovery time is desirable for real-time fault-tolerant systems. By handling failures within a pre-defined timing boundary, Time-To-Recovery can be bounded, and the system can operate continuously. To limit Time-To-Recovery, our previous research [9] categorized software tasks into three classes: Hard Recovery Tasks, Soft Recovery Tasks, and Best-Effort Recovery Tasks. We apply the same classification to Software-Components (SW-Cs), where a runnable is a part of an

SW-C [6]. An SW-C with the requirement of completing a released job within a D_i units of time, even with the presence of failures, is classified as a *Hard Recovery Software-Component* (HSC). A *Soft Recovery Software-Component* (SSC) is an SW-C with a more relaxed recovery-time requirement; a released job can miss the deadline D_i, but it should be recovered within a bounded time when a failure is present. An optional SW-C that is not critical for system operation and does not require bounded recovery time is classified as a *Best-Effort Recovery Software-Component* (BSC). The R-BATCH (Reliable Bin-packing Algorithm for Tasks with Cold standby and Hot standby) scheme [9] provides a comprehensive solution that allocates HSC, SSC, and BSC to multi-processors for guaranteeing reliability requirements with bounded recovery times.

From a dependability perspective, a single failure of a runnable within an application flow may affect all of its successors such that the overall application flow requirement is violated. R-BATCH, which uses *Hot Standby* and *Cold Standby* with stand-alone runnables, cannot be directly utilized for systems with data dependencies. Therefore, we propose a new allocation algorithm, R-FLOW (Reliable application-FLOW-aware SW-C partitioning algorithm), designed for the application flow model. R-FLOW has three properties that distinguish it from R-BATCH:

- A new application flow model that captures communication among SW-Cs,
- Clustering of SW-Cs based on their communication bandwidth needs,
- Controlling the number of Hot Standbys and Cold Standbys while guaranteeing the recovery-time requirement of all application flows.

In this paper, we assume a fail-stop failure model [5], where a failed component is assumed to stop generating any data and a working component can assume control by detecting the lack of output from the failed component. The component that takes over then aims to meet the desired deadline of the failed component.

We also aim at supporting R-FLOW within the AUTOSAR framework [2] for providing dependability. The AUTOSAR framework comprises of application software, a *Virtual Functional Bus* and a *Runtime Environment* (RTE). The RTE is responsible for enabling interaction between application software and the operating system along with support for different services within AUTOSAR. Currently, there is no explicit support for recovering from task failures by means of task replications within AUTOSAR. The standard instead assumes that architecture designers will introduce custom extensions to meet such reliability needs. In this paper, we propose enhancements to the different layers of AUTOSAR to enable fault-tolerance and, therefore, provide support for R-FLOW. This enables fault-tolerance support to be built into the framework by providing an API for fault-tolerance rather than having to rely on custom service modules.

The rest of this paper is organized as follows. The next section describes the system model with the timing properties of different SW-C replication mechanisms in a multiprocessor environment. Then, R-FLOW, a new SW-C partitioning algorithm, is proposed. Based on the proposed algorithm, R-FLOW, fault-tolerance characteristics within the AUTOSAR framework is summarized. After that, R-FLOW is evaluated by using an AUTOSAR-compliant fault-tolerant platform implementation. Finally, we provide our concluding remarks in the final section.

SYSTEM MODEL AND DESIGN

One of our goals is to minimize the required number of processors for allocating a given set of AUTOSAR runnables with data dependencies while guaranteeing end-to-end delays of all application flows even if there are permanent (fail-stop) processor failures. Given a system reliability requirement, a certain number of failures will be tolerated. We now present the software/hardware architecture and hardware fault model.

SOFTWARE ARCHITECTURE

We assume a set of given runnables, Υ, which is composed of n runnables, $\rho_1, \rho_2, \dots$, and ρ_n. Each runnable ρ_i is a part of an atomic SW-C, ω_j, which may have several runnables. For representing the relationship between a runnable, ρ_i, and an SW-C, ω_j, we define a function Θ such that $\Theta(\rho_i) = \omega_j$ when ω_j contains ρ_i. The inverse function of Θ, Θ^{-1}, returns all runnables contained in an SW-C. The set of SW-Cs, Ω, is also given. For guaranteeing different recovery requirements, we classify Ω into three overlapping sets, *Hard Recovery Software Component* Set, Ω_H, *Soft Recovery Software Component* Set, Ω_S, and *Best-effort Recovery Software Component*

Set, Ω_B. Each SW-C, ω_j, is an element of at least one[1] of the three subsets, Ω_H, Ω_S, and Ω_B. The exact definition of these subsets is defined in a later section.

A subset $A_k \subset \Upsilon$ contains runnables for the k^{th} application flow out of total m application flows. For a certain runnable, $\rho_i \in A_k$, ρ_i also can be an element of A_l, where $k \neq l$. The relationship among runnables in the k^{th} application, A_k, is represented by a directed graph, G_k. Inside the graph G_k, a node u denotes a runnable, $\rho_i \in \Upsilon$, and an edge (u, v) of G_k indicates a data flow from u to v. An edge (u, v) has its own message, m_{uv}, which is generated by the node u, and consumed by the node v in G_k. The allocation of each runnable, ρ_i, to G_k is assumed to be given at design time. Let $u(G_k, \rho_i)$ be the node of G_k allocated to the runnable, ρ_i. If ρ_i is not a member of A_k, the value of $u(G_k, \rho_i)$ is $\emptyset$. This function also works with an SW-C. An application flow, A_k, is represented by a couple (Δ_k, T_k^A) where, Δ_k is an end-to-end delay requirement, and T_k^A is a period of the application A_k. The end-to-end delay is defined as the worst-case delay between the release time of the first executed node in G_k and the completion time of the last executed node in G_k. All runnables in the application A_k share the same period T_k^A.

A runnable, ρ_i, is represented by a quadruple $(C_i, T_i, D_i, \alpha_i)$, where C_i is its worst-case execution time, T_i is its period, D_i is its relative deadline to the release time of each job (instance), and α_i is the ratio of recovery time to relative deadline. The recovery time is defined as the time instant relative to the release time of a failed job, and the failed job must be fully recovered at the recovery time. For instance, if $\alpha_i = 1$, the replica of the runnable ρ_i should recover what ρ_i is supposed to execute within the original relative deadline, D_i. Let R_i denote the response time of a runnable ρ_i, where the response time is the time interval between job release and job completion of ρ_i. Although all runnables in one application have the same period, the deadlines of those runnables will be determined based on the end-to-end delay requirement, Δ_k for A_k.

An SW-C ω_j is also represented by a quadruple $(C_j, T_j, D_j, \alpha_j)$, but it differs from a runnable in that C_j is set to $\max_{\forall \rho_i \in \Theta^{-1}(\omega_j)} C_i$, T_j is $\min_{\forall \rho_i \in \Theta^{-1}(\omega_j)} T_i$, D_j is $\min_{\forall \rho_i \in \Theta^{-1}(\omega_j)} D_i$, and α_j is $\min_{\forall \rho_i \in \Theta^{-1}(\omega_j)} \alpha_i$. Let u_j denote the utilization of an SW-C ω_j, and it is defined as ${C_j}/{T_j}$. The density of ω_j is defined as ${C_j}/{D_j}$ and denoted by d_j. Both u_j and d_j are used for measuring the amount of processor resources consumed by ω_j. Every SW-C ω_j can have $\psi(j)$ Hot Standby replicas[2], which is represented by $\omega_{j,1}^h, \omega_{j,2}^h, \dots, \omega_{j,\psi(j)}^h$. Either ω_j or $\omega_{j,0}^h$ denotes the primary of ω_j. Since our objective is to tolerate π processor failures, each SW-C ω_j also can have $\zeta(j)$, which is $\pi + 1 - \psi(j)$ Cold Standby replicas, which are represented by $\omega_{j,1}^c, \omega_{j,2}^c, \dots, \omega_{j,\zeta(j)}^c$.

HARDWARE ARCHITECTURE AND FAULT HYPOTHESIS

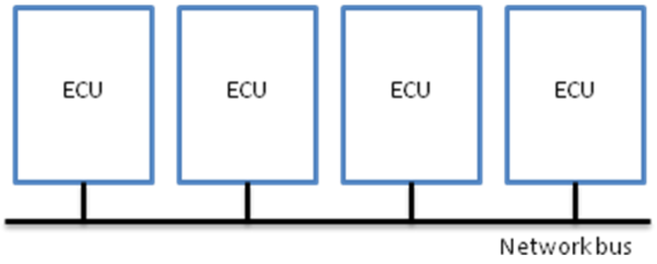

Figure 3: Abstracted hardware architecture

We will adopt the abstracted platform architecture shown in Figure 3. In this architecture, we will assume that we use a fault-tolerant network such as FlexRay [4] as the underlying in-vehicle network. Leveraging timeliness of such a network can guarantee the bounded delivery of packets generated by each SW-C. In other words, we can focus only on permanent processor failures rather than network failures.

[1] If an SW-C is involved in both Ω_H and Ω_S, the SW-C has runnables of both Hot Standby and Cold Standby.

[2] In this paper, we create a new AUTOSAR task for each replica of SW-C. The reason behind this is that assigning several runnables to one AUTOSAR task can delay the Time-To-Recovery when a failure occurs.

A set of processors (or ECUs), P, is used for running a runnable set, Υ. P is composed of l homogeneous processors, $P_1, P_2, \ldots, P_l$. Then, Π_i is the set of processors utilized by SW-C ω_i and its Hot Standby replicas. Each element of Π_i, $\Pi_{i,j}$ is the processor allocated to $\omega_{i,j}$, the j^{th} Hot Standby replica of SW-C ω_i. Two replicas of the same SW-C cannot be allocated to the same processor. We refer to this as a *placement constraint*. It can be expressed as $\forall i,\ \Pi_{i,j} \neq \Pi_{i,k}$, where $j \neq k$. Each processor P_k has its own failure rate, f_k, which denotes the probability of a permanent failure. We assume homogeneous processors for convenience of presentation, i.e. $f_1 = f_2 = \cdots = f_l = f$. In this paper, only permanent failures are considered [5]. Occurrences of fail-stop failures of these processors can be detected by using periodic heartbeat signals.

DEALING WITH END-TO-END DELAY OF AN APPLICATION FLOW

Under the given requirements on end-to-end delay of an application flow and DMS (Deadline Monotonic Scheduling) [18], the deadline of a runnable should be determined. The shorter the deadline of a runnable, the more responsive is the system. However, with a shorter deadline, a runnable has larger density, and the system may need more processors. Hence, our design objective is to pick the longest deadline that meets the end-to-end delay of a given application flow.

The end-to-end delay for an application flow is calculated in [10], where a pipeline task model is used on a FlexRay network. For A_k, Δ_k is bounded by the following equation.

$$\Delta_k \leq \sum_{i=1}^{|A_k|-1} \left(R_{u(G_k,\rho_i)} + \left\lceil \frac{T_{m_{u(G_k,\rho_i)u(G_k,\rho_{i+1})}}}{\gamma} \right\rceil \times \gamma + T_{u(G_k,\rho_{i+1})} \right)$$

$$+(|A_k| - 1) \times \varphi + R_{u\left(G_k,\rho_{|A_k|}\right)} = \Delta_k^B \qquad (1)$$

where, $|A_k|$ represents the number of elements in A_k, R_ρ is the response time of a runnable ρ, T_m is the period of message m, γ is the communication cycle length of the FlexRay network, T_ρ is the period of runnable ρ, and φ is the duration of a static slot in the FlexRay network. Under the assumption that the FlexRay network is synchronized among ECUs, $T_{\rho_{i+1}}$ can be omitted because $T_{\rho_{i+1}}$ reflects the offset to ρ_i in terms of ρ_{i+1}. The response time of each runnable can be obtained by using the standard response-time test in the AUTOSAR framework as described in [14]. Then, we can obtain the deadlines of runnables by satisfying the following objective function for each application flow.

$$\begin{aligned} &\text{minimize}_{\forall \rho_i \in A_k} \textstyle\sum_{i=1}^{|A_k|} d_i && (2)\\ &s.t.\ \Delta_k \leq \Delta_k^B && (3)\\ &\quad C_i \leq D_i \text{ for } \forall i && (4) \end{aligned}$$

In the above objective function, we want to minimize the density d_i, which is defined as C_i/D_i, for saving resources while the end-to-end delay requirement is met by Constraint (3). If we find a set of deadlines for each runnable which minimizes the objective function above, those deadlines can be used for allocating runnables to processors. In this paper, due to the NP-hardness of the given objective function, we use a heuristic which assigns a deadline to each runnable that is proportional to its period such that it follows the RMS (Rate Monotonic Scheduling) priority assignment [13]. Each deadline is assigned by the following equation.

$$D_i = \left(\Delta_k - \sum_{\forall \rho_j \in A_k} \left\lceil \frac{T_{m_{u(G_k,\rho_j)u(G_k,\rho_{j+1})}}}{\gamma} \right\rceil \times \gamma \right) \times T_i \Big/ \sum_{\forall \rho_j \in A_k} T_j \qquad (5)$$

HOW TO DETERMINE STANDBY TYPE

Task replication is a fundamental method for improving the reliability of a target system. In that sense, replicating SW-Cs in the AUTOSAR framework can increase system reliability. SW-Cs on multiple ECUs can recover SW-Cs on failed processors. In this paper, we consider two techniques for replicating SW-Cs viz. Hot Standby and Cold Standby with different timing characteristics [9].

- Hot Standby Approach: This approach uses two or more on-line copies of a certain SW-C. One or more replicas of the SW-C will be active simultaneously. Each replica must be on a different processor, in order to make sure that at least one of them is working when an ECU failure occurs. When a failure occurs, a replica will take over the task on a failed processor. At least one of the backup replicas should meet the original deadline when a failure occurs. Hot Standby replicas are released synchronously with that of the primary copy and execute in parallel with the same deadline. Any Hot Standby can be promoted to be the primary after a primary fails.
- Cold Standby Approach: In this case, replicas are not active until triggered due to failures. In other words, they only utilize memory, but not processor cycles under normal operation, and they are activated on demand when failures occur. Only the state information of each primary copy needs to be shared and updated among replicas. When failures occur, they should be detected as soon as possible, and the SW-Cs on failed ECUs should be recovered using replicas within a predefined time. The main benefit of using Cold Standbys is that the Cold Standby replicas for processors other than those hosting a certain SW-C can be consolidated. Suppose an SW-C set $\Omega = \{\omega_1, \omega_2\}$, and both SW-Cs have a utilization of 0.6 each. They cannot fit into one processor together. Hence, in order to tolerate one failure per SW-C by using only Hot Standbys, two more processors will be required. A Cold Standby, however, can use only one processor since the Cold Standbys for both ω_1 and ω_2 can be co-resident on a single processor if their primaries are running on different processors.

The benefit of using a Hot Standby is its ability, when the primary fails, to meet the original deadline with a smaller timing penalty than the Cold Standby approach, since all Hot Standbys are running concurrently with the primary component. However, Hot Standbys require additional resources, $\psi(j)$ times u_j for ω_j. For the Cold Standby approach, when there are no failures, only the primary copy is executed. Since processor failures can be detected through the properties of fail-stop processors, the failure of a primary copy triggers the execution of a Cold Standby unless a Hot Standby is running. For bounding recovery time, we use *Transient Overload Density* (TOD), which is extended from *Transient Overload Utilization* defined in [9]. TOD is the additional processor utilization required for an SW-C that is recovered through the Cold Standby approach. TOD is the required additional resource for re-executing an SW-C on a new processor within the remaining time, $\alpha_j D_j - R_j$, after a processor $\Pi_{j,0}$ fails. Let d_j^t denote the TOD of ω_j. Then, we can use the result that $d_j^t \leq d_j$, when $\alpha_j \geq 2$ from [9] to determine the type of replicas for ω_j.

By using a large α_j, d_j^t can be relaxed. The importance of this result is seen in the following example. Suppose that an SW-C ω_j with $\alpha_j = 1$ uses a Standby. Then, the backup copy of ω_j should meet the condition $D_j - R_j \geq C_j$. Therefore, $d_j^t = \frac{C_j}{D_j - R_j}$. Since $D_j - R_j < D_j$, $d_j^t \geq d_j$, where the Hot Standby approach might be appropriate. For $\alpha_j > 1$, the utilization of the standby goes down, but the recovery time goes up. Hence, based on the value of α_j, we can choose the type of Standby. An important observation regarding the Cold Standby approach is that the processor running the backup should have enough unused utilization. The reserved slack for Cold Standby replicas can be used by any SW-C in the presence of failures, but the reserved utilization for Hot Standby cannot be utilized by other SW-Cs.

A Cold Standby can also be promoted to a Hot Standby if the primary of ω_j fails and the current number of ω_j is less than $\psi(j)$. Since at least one Hot Standby for a certain ω_j can meet its original deadline, D_j, the system can be ready to tolerate another potential failure if a Cold Standby can be promoted to a Hot Standby. By using this approach, we can tolerate as many as π failures for ω_j.

FAULT-TOLERANT SW-C ALLOCATION WITH APPLICATION FLOWS

This paper proposes a comprehensive method for allocating SW-Cs[3] to processors while meeting the requirements on reliability and end-to-end delay. The proposed scheme is composed of two complementary phases: flow-aware allocation and reliability-aware allocation. The flow-aware allocation tries to co-locate SW-Cs which have dependencies on each other such that the end-to-end delay can be reduced. After all primary SW-Cs are allocated, their replicas can be placed on appropriate processors while satisfying the placement constraint. Hot Standbys and Cold Standbys will be allocated differently because Cold Standbys only use up memory, but not processor utilization.

[3] Allocating a runnable in the AUTOSAR framework implies that the corresponding SW-C is also allocated. In other words, all runnables of a SW-C should be assigned to one processor. The replication of a runnable therefore implies a SW-C replication.

SW-C ALLOCATION WITH APPLICATION FLOWS

Using fewer processors than conventional allocation methods is a major goal in this paper. The allocation of SW-Cs to processors during design time is a well-known bin-packing problem [15]. Each SW-C ω_j is treated as an item to be packed with a size, utilization value u_j, and these items will fill up one or more processors, each having a total capacity of 1 under the assumption that the SW-C periods are harmonic. If an item does not fit into the remaining space of any available processors, one more processor is added. Since the bin-packing problem is known to be NP-hard [15], there are various types of heuristics such as BFD (Best-Fit Decreasing), FFD (First-Fit Decreasing), WFD (Worst-Fit Decreasing), and NFD (Next-Fit Decreasing). However, none of these heuristics considers dependencies among SW-Cs for using a fewer number of processors.

In the previous section, we stated that co-locating SW-Cs that have dependencies on each other can reduce the amount of required resources. Suppose that we are given an application A, which is composed of a set of runnables, $\varUpsilon: \{\rho_1,\ \rho_2,\ \rho_3\}$. The graph G for A depicted in Figure 4, shows that A has a pipeline task model. Each runnable has the period of $100ms$, $100ms$, and $100ms$, and the worst-case execution time of $10ms$, $10ms$, and $10ms$, respectively. A set of SW-Cs corresponding to $\varUpsilon$ is given in $\Omega: \{\omega_1,\ \omega_2,\ \omega_3\}$, where each SW-C has only one runnable. The length of the communication cycle in FlexRay is assumed to be $20ms$, and the slot duration is $10\mu s$, which is negligibly small. The end-to-end delay $\varDelta$ for A should be equal to or less than $300ms$. Suppose that all three SW-Cs communicate via a FlexRay network. Then, if all job instances are completed by their deadlines, the given end-to-end delay cannot be satisfied due to the communication delay. For example, if a job instant of ω_1 which is released at $0ms$ finishes at $100ms$, a generated message by ω_1 will spend one FlexRay slot at least for transmission and may arrive at $120ms$ due to the communication cycle, where all these effects are reflected in Equation (5). If the same behavior happens on the second processor running ω_2, the end-to-end delay will not be satisfied. Therefore, in order to meet the requirement, the relative deadline of each SW-C should be $80ms$, $80ms$, and $80ms$, which gives a large density value, 0.375 ($^1/_8 + {}^1/_8 + {}^1/_8$). If we assume, however, that all SW-Cs are allocated to the same processor, a deadline of $100ms$ would be enough for each SW-C giving 0.3 as the total density of the given SW-C set, representing a utilization savings of 25%. This example illustrates that co-locating SW-Cs communicating with each other on a FlexRay network can save a substantial amount of resources.

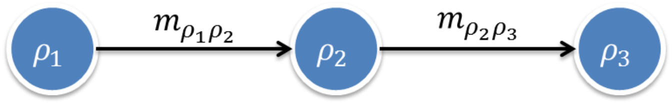

Figure 4: the graph G for the application A

We propose a technique called FBFD (Flow-BFD), a variant of BFD which considers dependencies among application flows. We use BFD as a base-line algorithm rather than other heuristics such as WFD and NFD due to its well-known worst-case behavior [15]. For SW-C allocation, the original BFD (1) sorts the SW-Cs in descending order of their densities, (2) allocates the next SW-C into the processor that it best fits into, (3) adds a new processor if an SW-C does not fit into any current processor, and (4) iterates this procedure until no SW-Cs remain. Here, we used SW-Cs instead of using runnables because $\forall \rho_i \in \omega_j$ should be allocated to a processor together.

FBFD uses a flexible definition of items to be packed. It tries to allocate all corresponding SW-Cs of an application flow as a single item when possible. Else, it splits these consolidated items when necessary. FBFD starts by combining SW-Cs as part of the same application flow, where these composite-SW-Cs can include several application flows because one SW-C can be used by several application flows. These consolidated SW-Cs are sorted in descending order of size in terms of their total densities. Then, FBFD fits the next SW-C into the best processor. If there is no processor into which an SW-C fits into, this SW-C is set aside and FBFD searches for any unallocated SW-Cs in the list which can fit into the current processors. If FBFD cannot find any such SW-C, it picks the biggest unallocated composite-SW-C among remaining composite-SW-Cs, and splits it into two pieces such that at least one piece can be allocated to the remaining space. In this case, the sum of densities of two pieces will be greater than the size of the original combined SW-Cs due to the communication delay. A new processor is added if necessary, and the remaining piece will be sorted again in descending order of sizes with other remaining unallocated composite-SW-Cs. These steps will be iterated until no SW-Cs remain. This procedure is also described in Figure 5.

```
FBFD ($\boldsymbol{\Omega}$: {$\boldsymbol{\omega_1}, \boldsymbol{\omega_2}, \ldots, \boldsymbol{\omega_n}$}, $\boldsymbol{\Pi}$: {$\boldsymbol{\Pi_1}, \boldsymbol{\Pi_2}, \ldots, \boldsymbol{\Pi_n}$}, $\boldsymbol{P}$)

$\Omega^C \leftarrow \emptyset$
// Consolidate SW-Cs based on application flows
for i=1 to n do
  if $\omega_i \notin \Omega^C$
    Find a composite-SW-C $\omega_j^c$ communicating with $\omega_i$
    if $\omega_j^c$ exists then
      $\Omega_j^C \leftarrow \Omega_j^C \cup \{\omega_i\}$
    else // Generate a new composite-SW-C
      $\Omega_{n(\Omega^C)}^C \leftarrow \{\omega_i\}$
      $\Omega^C \leftarrow \Omega^C \cup \Omega_{1+n(\Omega^C)}^C$
end for
Sort $\Omega^C$ in descending order of density
i ← 1
while($\Omega^C \neq \emptyset$)
  if i = 0 then,
    // Split the biggest composite into two pieces
    // such that one piece can fit into the processor
    // which has the largest remaining space
    // and satisfies the placement constraint
    $\Omega^C \leftarrow \Omega^C - \Omega_j^C + \Omega_{1+n(\Omega^C)}^C$
    Update $\Pi_j$ such that $\forall\omega_j \in \Omega_j^C - \Omega_{1+n(\Omega^C)}^C$
    Add a new processor, $P_{|P|}$
  // Satisfying the placement constraint
  For $\Omega_i^C$, find a best processor, $P_k$ s.t. $P_k \notin \Pi_j$ & $\forall\omega_j \in \Omega_i^C$
  if $P_k$ exists then,
    // Allocate $\omega_j^c$ to $P_k$
    Update $\Pi_j$ such that $\forall\omega_j \in \Omega_i^C$
    $\Omega^C \leftarrow \Omega^C - \Omega_i^C$
  else
    continue
  n ← n + 1
  n ← n mod $|\Omega^C|$
end while
return (P, Π)
```

Figure 5: Pseudo-code of FBFD

FAULT-TOLERANT SW-C ALLOCATION

We now extend FBFD to make it into a reliability-aware allocation scheme, R-FLOW. For achieving this goal, we (1) allocate replicas of SW-Cs and (2) spread those replications across different processors while satisfying the placement constraint. Different forms of replicas, Hot Standbys and Cold Standbys, will be allocated depending on the application flow properties.

Satisfying Reliability Requirements

Consider a uniform multiprocessor system with l processors. Let F denote the system Safety Integrity Level (SIL) [20] requirement specified in terms of the PFD (Probability of Failure on Demand). Let the reliability specification of each individual processor be f, denoting that the processors are designed to have a PFD less than f. Based on F and f, the system designer must estimate π, which is the minimum number of additional processors required to satisfy the system reliability. π should be greater than the maximum number (m) of processor failures that can be expected in $(m + \pi)$ processors.

$$\pi = min_p\{p \in Z \mid F \geq \sum_{j=p}^{m+p} Prob(exactly\ j\ failures)\} \quad (6)$$

$$\pi = min_p\left\{p \in Z \mid F \geq \sum_{j=p}^{m+p} \binom{m+p}{j} f^j(1-f)^{m+p-j}\right\} \quad (7)$$

A system designer may choose a value of π greater than the one obtained using Equation (6, 7) depending on design margins.

R-FLOW: Allocating Hot standby/Cold Standby with FBFD

Suppose we have the same exemplary set of SW-Cs as given in the previous subsection. The only difference is that ω_3 has one Hot Standby, $\omega_{3,1}^h$. Even if all the primary SW-Cs are allocated together by FBFD, the placement constraint brings a new processor for allocating $\omega_{3,1}^h$. The deadline of $\omega_{3,1}^h$ should be recalculated in this case because the pipeline, $\omega_1 \rightarrow \omega_2 \rightarrow \omega_{3,1}^h$, from Figure 4 should meet the end-to-end delay requirement. Since the periods of T_1 and T_2 are already known to be 100*ms* and 100*ms,* respectively, Equation (5) can be used for determining D_3 as 75*ms*, which generates a big item in terms of density. As this example shows, the deadline of each Hot Standby should be recalculated for allocating it. This deadline recalculation also affects the sorting which happens at the beginning of the allocation because an SW-C with low density does not necessarily mean a Hot Standby with low density. Therefore, after all primaries are allocated, the SW-Cs are sorted again in decreasing order, in accordance with BFD. This procedure of deadline recalculation and resorting is also executed whenever all *j*th replicas are allocated. Due to the recalculation of deadlines, the TOD of each SW-C is also affected. If the TOD of an SW-C becomes negative and it does not have any Hot Standby, the SW-C cannot be recovered within its required recovery time. In this case, the number of Hot Standby should be adjusted accordingly. Reflecting these properties, we propose a new allocation method, R-FLOW, which allocates primaries, Hot Standbys, and Cold Standbys with application flows. The pseudo-code for R-FLOW is described in Figure 6, where *generateVirtualTask()* is described in [9].

```
R-FLOW

// Allocate the primaries first
$(P, \Pi) \leftarrow FBFD(\Omega \leftarrow \{\omega_1, \omega_2, \dots, \omega_n\}, \Pi \leftarrow \emptyset, P \leftarrow \emptyset)$
// Allocate the replicas one by one
for j=1 to $\max_{\forall \omega_k \in \Omega}(\psi(k))$
  Recalculate the deadlines
  Recalculate the number of Hot Standbys
  // ignore SW-Cs that do not need j^th replicas
  $\forall \omega_i$ s.t. $\psi(i) < j, \omega_{i,j}^h \leftarrow \emptyset$
  $(P, \Pi) \leftarrow FBFD(\Omega \leftarrow \{\omega_{1,j}^h, \omega_{2,j}^h, \dots, \omega_{n,j}^h\}, \Pi, P)$
end for
$(\Omega, \Pi) \leftarrow generateVirtualTask(\Omega, \Pi, P, \pi)$
$(P, \Pi) \leftarrow FBFD(\Omega, \Pi, P)$
Return $(P, \Pi)$
```

Figure 6: Pseudo-code of R-FLOW

FAULT-TOLERANCE WITH AUTOSAR

We now describe how an implementation of our approach can be integrated into the AUTOSAR framework. We made modifications to various modules within AUTOSAR to enable our fault-tolerance algorithm. This section gives a description of the changes made as well as a description of services introduced as part of the implementation.

To support the replication of SW-Cs, the AUTOSAR Software Component Template was modified to introduce new properties. An Automotive Safety Integrity Level (ASIL) value is assigned to every SW-C to represent the level of safety required for that particular component. This enables R-FLOW to pick the replication scheme to apply, assuming that the replication of an SW-C results in the replication of all of its runnables. A new structure was added to the Software Component Template which provides a description of the internal data representing the current state of the runnable. This is required to enable Cold Standbys to remain synchronized with the primary component to ensure that the Cold Standby has the most current state available when it is activated. To this end, a property describing the maximum initialization time of the SW-C is added which describes the amount of time needed for the Cold Standby to produce valid data.

A health status module was added to the AUTOSAR ECU Specification and it is responsible for sending the ECU status to all other ECUs. This is relayed through the AUTOSAR Communication Service (COM) which is responsible for communicating the health status to all ECUs. The appropriate mechanisms for COM are put in place by introducing a health message which is broadcasted using the existing COM API. Several callbacks were added to the AUTOSAR Runtime Environment (RTE). One of them is a callback function from the COM module regarding the health status of all ECUs. This callback function is responsible for activating any replicas that reside on this ECU depending on the status of other ECUs and as part of the on-line procedures described in the previous section.

There are several assumptions regarding offline analysis and synthesis. First of all, it is assumed that all runnables within SW-Cs are executed periodically within the context of an AUTOSAR Task. Secondly, the runnable to Task Mapping exists before R-FLOW is used for replication and allocation. Lastly, the ECU description is assumed to be already available as part of the RTE Generation process. This provides R-FLOW with the description of available resources for task allocation. An example depicting how replication is done within the System Model is shown in Figure 7. Here, we have three SW-Cs S1, S2, and S3, two AUTOSAR Tasks T1 and T2, and two ECUs E1 and E2 which are connected through a network. Suppose that R-FLOW decides to replicate S1, giving component S1′. This results in the replication of its runnables R1_S1 and R2_S1, producing R1_S1′ and R2_S1′ respectively. Given that S1′ is mapped to E2 as per R-FLOW, we now have to re-schedule the Tasks that need to run on the ECUs. In this paper, new Tasks will be created as mentioned in the previous section. This action can be automated by a tool using predefined rules such as creating a new task for each replicated runnable. Tasks can contain multiple runnables to capture any explicit ordering present as part of a task schedule on the original ECU. In this example, a new Task T3 is created that contains the replicated runnables. The new configuration is depicted in Figure 8.

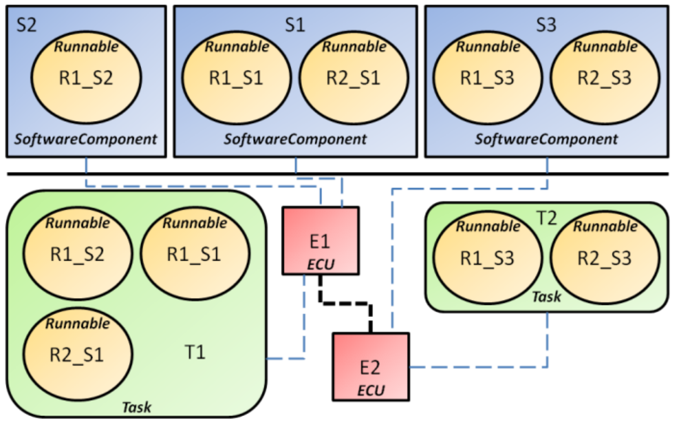

Figure 7: Example Configuration without Replication

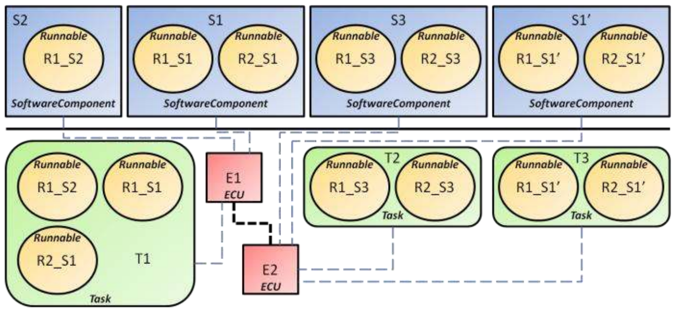

Figure 8: Example Configuration with Replication

IMPLEMENTATION

We developed an experimental platform to evaluate our approach and look at overheads and costs associated with fault-tolerance. A real automotive platform was necessary to show the complexities involved in adding fault-tolerance to the system. This section describes the system architecture in detail.

HARDWARE

The hardware architecture was built as part of a testbench as shown in Figure 13. The computational architecture is comprised of five Softec HCS12X development kits using the MC9S12XDP512 processor from Freescale [7]. The ECUs run at 50Mhz and use a 12V supply. Daughter boards from Freescale are used for FlexRay connectivity. The ECUs are connected using a dual-channel FlexRay bus and two Full-Speed CAN networks. One of the five ECUs acts as the system gateway and is connected to a PC using two RS232 channels. This ECU acts as the fault injection module and is also used to collect data from the system for diagnosis and analysis. The types of faults that can be injected include communication shutdown, shorting of bus channels, shutdown and restart of ECUs, and injection of faulty network bus messages. These faults can be controlled using a PC interface and data collected by the system are analyzed at the PC as well.

SOFTWARE

The basic software running on the hardware includes the RTA-OSEK [8] operating system and generated code from SysWeaver [18] using an AUTOSAR Code Generation module that was added to SysWeaver. The code generated conforms to the AUTOSAR 4.0 specifications and produces the minimum required implementation to produce a working system. All the relevant modules including the RTE, COM, PDU Router and SW-C supplementary headers are generated. An OIL file for configuration of the RTA-OSEK OS is also generated for each ECU. This generated code includes the necessary code modifications required as part of the fault-tolerance support described in the previous section. Integration was added for R-FLOW within the *Fault-Tolerance View* of SysWeaver to produce the necessary replicas and SW-C allocation. Code is then generated for every ECU along with a configuration file for the Gateway ECU. The Freescale CodeWarrior compiler for the HCS12X is then used along with the RTA-OSEK configuration tool to produce an executable for each ECU. This is done automatically by SysWeaver.

The system model is created within SysWeaver and comprises of SW-Cs that were designed in the tool as well. The *Functional View* within SysWeaver comprises of communicating SW-Cs. Each SW-C has the relevant AUTOSAR properties associated with it, including the information required by R-FLOW. For this paper, we concentrate on *Sender-Receiver Communication* as the only communication mechanism between SW-Cs since a chain of communicating SW-Cs constitutes an application flow. The *Deployment View* within SysWeaver consists of the hardware configuration including the ECUs and any network buses within the system. Each ECU has properties associated with it as described in the AUTOSAR ECU Description. A *Dynamic View* exists which contains AUTOSAR Tasks, where each AUTOSAR Task contains a Schedule Table of runnables with their respective Task offset and periodic intervals. Given these properties, R-FLOW is then invoked to produce the required Replicas and Task Allocation. The Fault Tolerance View in SysWeaver shows the replicas produced along with the type of replication, and the SW-C allocation can be seen in the Deployment View. Figure 14 shows an example system model in SysWeaver. The figure shows 2 application flows, A & B.

RESULTS

In this section, we evaluate the performance of FBFD relative to BFD, which does not consider the effects of application flows. Then, we will show the benefits of using R-FLOW on randomly chosen application flows. We analyze the characteristics of these schemes by varying the number of application flows and the number of SW-Cs in an application flow. Our experiments pick different end-to-end delays: 500*ms*, 1000*ms*, 1500*ms*, or 2000*ms*. In order to get the period value for each SW-C within an application flow, we divide the end-to-end delays by the number of SW-Cs in the flow. Then, the worst-case execution time of each SW-C is randomly chosen such that the utilization of each SW-C varies between 0% and 30%. The number of application flows is itself varied from 10 to 20, and the number of SW-Cs in an application flow is varied from 10 to 15 in each experiment. In all our experiments, the communication cycle length is set at 20*ms* and each data point is averaged after 500 iterations.

The performance metric used for comparison is the ratio of saved processors which is defined as $n(BFD) - n(FBFD) \big/ n(BFD)$,

where $n(A)$ means the number of required processors when scheme A is used. Higher the value of this metric, better is the performance of the scheme under consideration.

For R-FLOW, while the same parameter variations above are applied, we also conduct experiments on using only Hot Standbys or only Cold Standbys for guaranteeing system reliability. We vary the number of tolerated failures from 1 to 4, yielding a total of 2 to 5 copies due to the inclusion of the primary.

Figure 9 and Figure 10 show the number of saved processors when FBFD is used, normalized to the number of processors required by BFD. Figure 9 depicts the ratio of processors saved when the number of application flows and the end-to-end delay are varied. The number of SW-Cs in each application is fixed at 10. As seen in Figure 9, FBFD can save a substantial number of processors (up to 45% processors) when the end-to-end delay is 500*ms*. FBFD can save more processors when the end-to-end delay is shorter because the overhead of communication on an application flow represents a larger ratio of the delay when the end-to-end delay is shorter. It can also be seen that the number of application flows does not affect the performance. This means that the size of an SW-C set does not play a major role. Figure 10 presents the results of an experiment where the number of SW-Cs in an application flow is varied from 10 to 15. Again, a large number of processors (up to 56% processors) can be saved when the end-to-end delay is 500*ms*. As shown in Figure 10, a larger number of stages in an application flow have a greater impact on the performance of this algorithm because of the bigger impact of communication delays on shorter end-to-end delays.

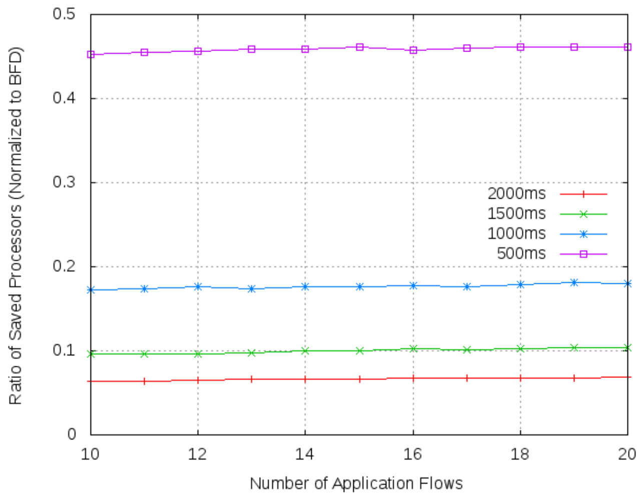

Figure 9: The ratio of saved processors when we use FBFD under varying application flows and fixed number of SW-Cs per application flow

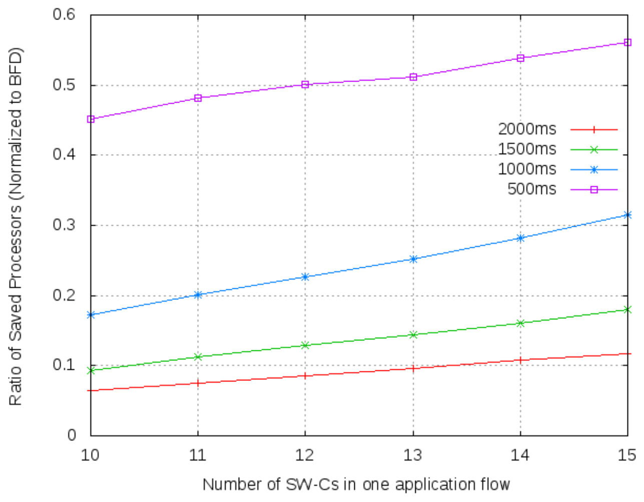

Figure 10: The ratio of saved processors when we use FBFD under fixed number of application flows and varying number of SW-Cs per application flow

Figure 11 and Figure 12 show the percentage of saved processors when R-FLOW is used, normalized to R-BATCH. In each of these experiments, the number of tolerated processor failures is varied from 0 to 4, where tolerating 0 processor failures is equivalent to

FBFD. The number of applications and the number of SW-Cs in an application flow are both fixed at 10. Figure 11 captures the results for the experiment when only Hot Standbys are used. As seen in the figure, R-FLOW can save up to about 60% of processors when the end-to-end delay is 500*ms*. The end-to-end delay is also the dominant performance factor when Hot Standbys are used. The rate of improved savings growth is slow because the density (the ratio of computation time to deadline) of a Hot Standby is different from that of the primary when R-FLOW is utilized. This is not true when R-BATCH is used. Since the size of a Hot Standby for R-FLOW is larger due to additional communication delays between the primary and the Hot Standby, the number of saved processors is not increased as more replicas are introduced to tolerate more failures. Figure 12 represents the case where, only Cold Standbys are used for recovering from processor failures, and has a different trend. The ratio of saved processors does not vary much as the end-to-end delays of application flows decrease. The reason behind this is that virtual SW-Cs recover several SW-Cs simultaneously and an SW-C with a higher density can save more. Therefore, the effect of end-to-end delay on the ratio of saved processors is negligible.

In summary, R-FLOW can save a substantial number of processors (up to 60% processors) relative to the required processors by the R-BATCH scheme.

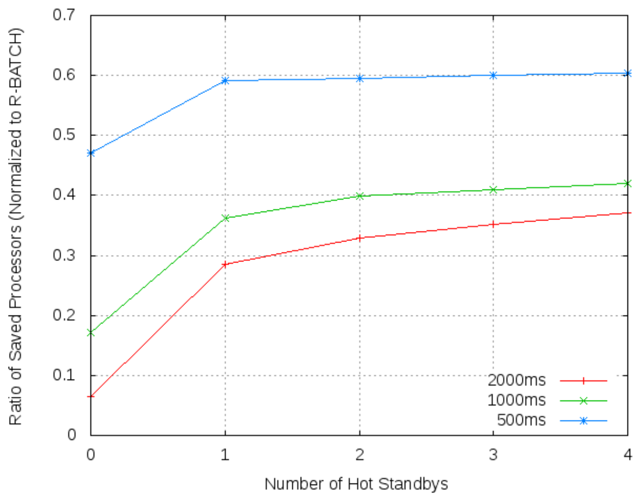

Figure 11: The ratio of saved processors when we use R-FLOW in order to tolerate varying number of processor failures with Hot Standby

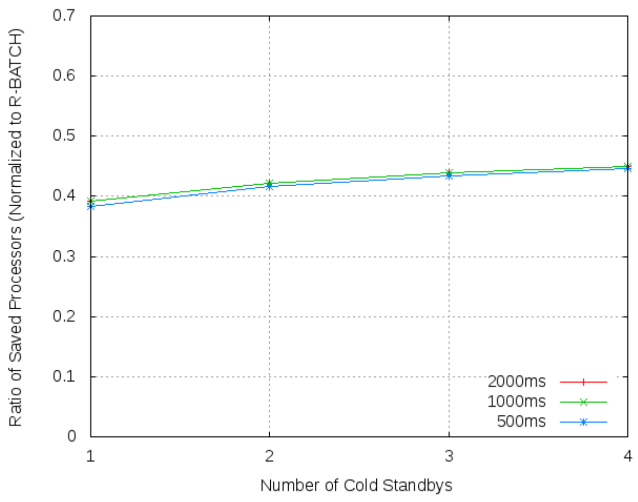

Figure 12: The ratio of saved processors when we use R-FLOW in order to tolerate varying number of processor failures with Cold Standby

SUMMARY/CONCLUSIONS

In this paper, we have proposed a processor assignment methodology called R-FLOW for allocating SW-Cs while the end-to-end delay of application flow and the given reliability requirement are guaranteed. We have defined a new model using an abstraction called an *application flow*, which enables timing analysis. We have also described the classification of SW-Cs based on their fault-tolerance requirements. The classification and application flow models are used within R-FLOW, an application flow-aware SW-C partitioning algorithm for improving system reliability. Our results have shown that R-FLOW results in a savings of up to 45% of processors when only primary components are allocated. If replicas are used to enhance reliability, savings of more than 60% of processors can be achieved as compared to our earlier scheme called R-BATCH, while satisfying the same level of reliability requirements. Finally, we have described how R-FLOW can be used within the AUTOSAR framework, and have implemented this algorithm within the SysWeaver tool from Carnegie Mellon resulting in automatic code generation for an AUTOSAR-compliant system.

As our next steps, we will focus more on improving the on-line performance of R-FLOW by introducing a new protocol, which is responsible for managing the primary, Hot Standbys, and Cold Standbys of each SW-C. We will also compare our approach to the methods defined as part of the ISO26262 standard [16] on Functional Safety, and investigate compliance with the requirements and implementation aspects of the standard.

REFERENCES

1. R.K. Jurgen, X-By-Wire Automotive Systems, SAE International, 2009.
2. “AUTOSAR,” Automotive Open System Architecture.
3. T.X. Mei, M. Shafik, R. Lewis, H. Walilay, M. Whitley, and D. Baker, “Fault Tolerant Actuation for Steer-by-Wire Applications,” Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on, 2007, pp. 1-8.
4. R. Belschner, J. Berwanger, C. Ebner, H. Eisele, S. Fluhrer, T. Forest, T. Fuhrer, F. Hartwich, B. Hedenetz, and R. Hugel, “FlexRay Requirements Specification,” FlexRay Consortium, Internet: http://www. flexray. com, Version, vol. 2, 2002.
5. D. Pradhan, Fault-tolerant computer system design, Prentice Hall PTR, 1996.
6. AUTOSAR, “Glossary V2.2.0 R4.0 Rev 1,” 2009
7. Freescale, “4310STARTERKIT Product Summary Page.”
8. ETAS, “ETAS - RTA-OSEK - RTA Software Products - Software Products & Systems - Product Search - ETAS Products,” 25T12:36:14+02:00. 2007.
9. J. Kim, K. Lakshmanan, R. Rajkumar, "R-BATCH: Task Partitioning for Fault-tolerant Multiprocessor Real-Time Systems," Proceedings of 10th IEEE International Conference on Computer and Information Technology (CIT), 2010
10. K. Lakshmanan, G. Bhatia and R. Rajkumar, “Integrated End-to-End Timing Analysis of Networked AUTOSAR-Compliant Systems,” Proceedings of the Design, Automation, and Test in Europe (DATE), 2010
11. A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” IEEE Transactions on Dependable and Secure Computing, pp. 11-33, 2004.
12. "Progress report No.2 on the accident on 1 June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro - Paris," BEA (Bureau d’Enquêtes et d’Analyses pour la sécurité de l’aviation civile), 2009.
13. C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” J. ACM, vol. 20, no. 1, pp. 46-61, 1973.
14. P. Hladik, A. Deplanche, S. Faucou, and Y. Trinquet, “Adequacy between AUTOSAR OS specification and real-time scheduling theory,” in International Symposium on Industrial Embedded Systems (SIES), 2007.
15. D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, “Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms,” SIAM Journal on Computing, vol. 3, no. 4, pp. 299-325, Dec. 1974.
16. International Organization for Standardization, “ISO/DIS 26262 – Road vehicles – Functional safety,” ISO Publications, 2009
17. C. Urmson et al. “Autonomous driving in urban environments: Boss and the urban challenge,” In The DARPA Urban Challenge, pages 1–59. 2009
18. D. de Niz, G. Bhatia, and R. Rajkumar, “Model-Based Development of Embedded Systems: The SysWeaver Approach”, Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2006
19. N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, “Hard Real-Time Scheduling: The Deadline-Monotonic Approach”, Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software, 1991
20. International Electrotechnical Commission, IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems, 65A/254/FDIS, IEC:1999.

CONTACT INFORMATION

Junsung Kim, Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

Gaurav Bhatia, Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

Ragunathan (Raj) Rajkumar, Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

Markus Jochim, General Motors Research & Development, Warren, MI, USA

ACKNOWLEDGMENTS

We would like to thank Tom Fuhrman from GM for helpful comments on this paper. We would like to thank Jonas Cleveland for working on the Power Wheels platform. We would also like to thank ETAS for providing us with evaluation licenses for RTA-OSEK.

DEFINITIONS/ABBREVIATIONS

AUTOSAR	AUTomotive Open System ARchitecture
BFD	Best Fit Decreasing
BSC	Best-Effort Recovery SW-C
CAN	Controller Area Network
COM	Communication Service
DMS	Deadline Monotonic Scheduling
ECU	Electronic Control Unit
FBFD	Flow-BFD
FFD	First Fit Decreasing
HSC	Hard Recovery SW-C
NFD	Next Fit Decreasing
PFD	Probability of Failure on Demand
R-BATCH	Reliable Bin-packing Algorithm for Tasks with Cold standby and Hot standby
R-FLOW	Reliable application-FLOW-aware SW-C partitioning algorithm
RTE	Run Time Environment
SBW	Steer-by-Wire
SIL	Safety Integrity Level
SSC	Soft Recovery SW-C
SW-C	Software-Component
TMR	Triple Modular Redundancy
TOD	Transient Overload Density
WFD	Worst Fit Decreasing

APPENDIX

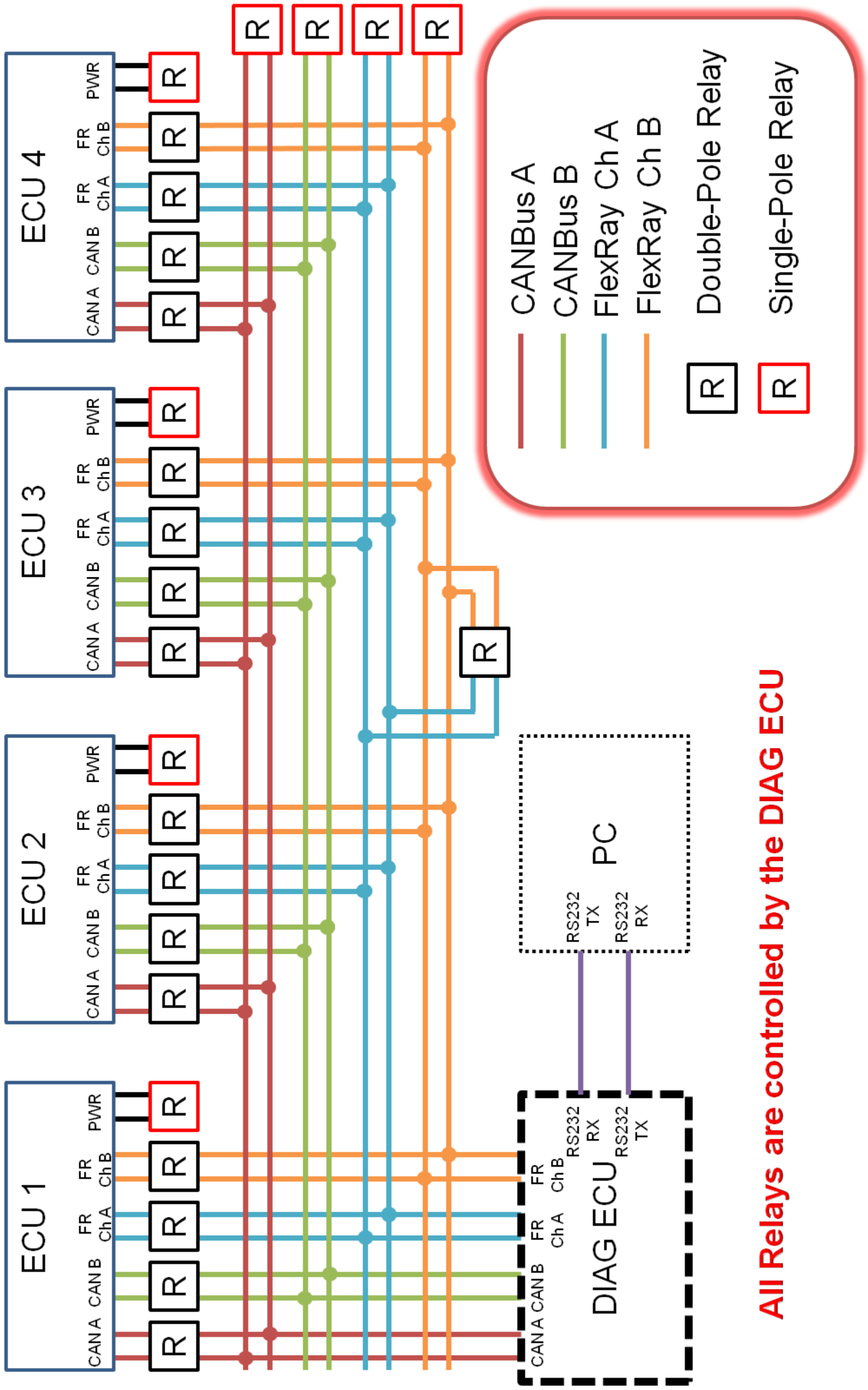

Figure 13: Testbench Architecture

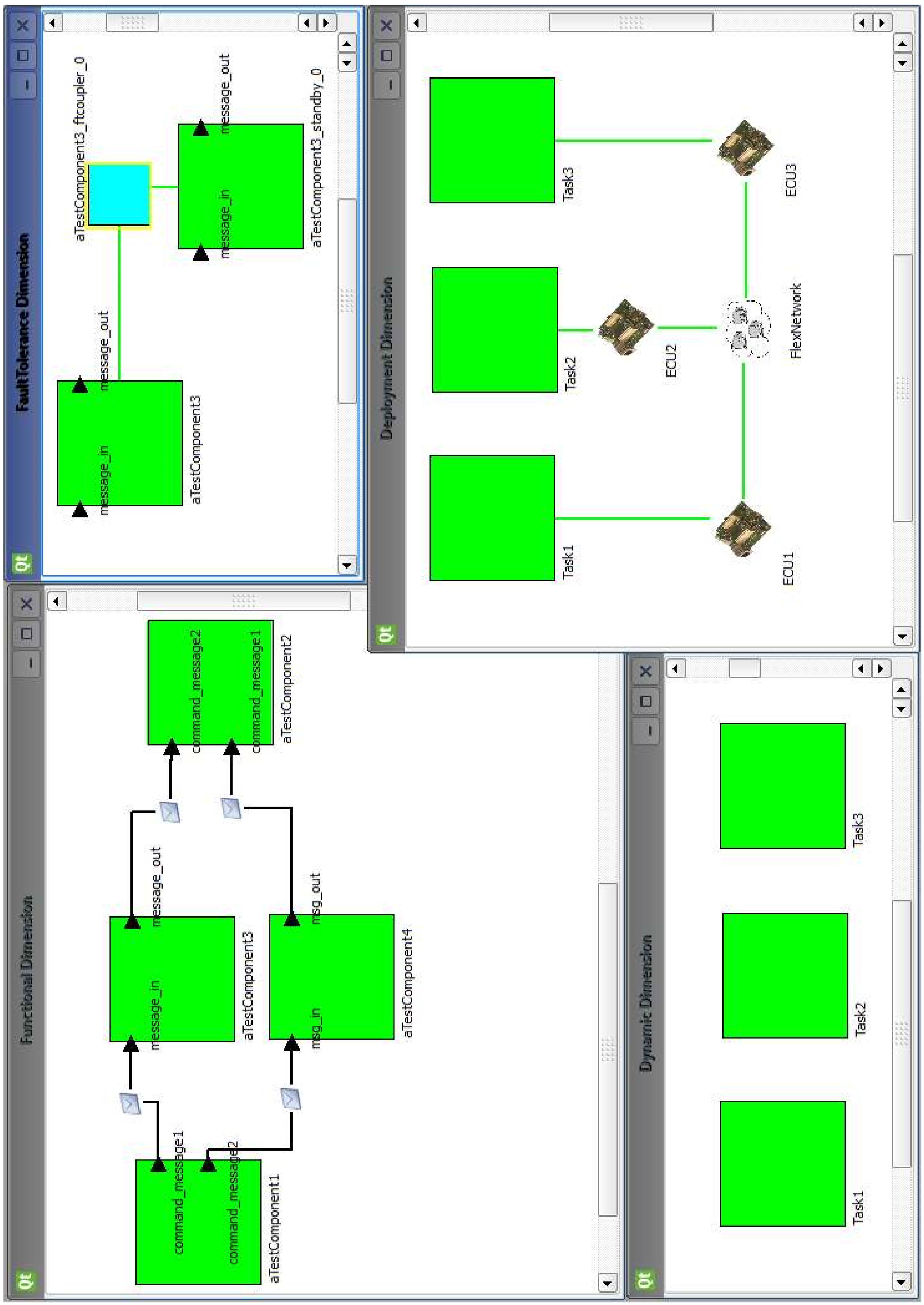

Figure 14: SysWeaver System model