Dependability Driven Integration of Mixed Criticality SW Components

Shariful Islam, Robert Lindström and Neeraj Suri
DEEDS Group, Department of Computer Science
Technische Universität Darmstadt, Germany
{ripon,rl,suri}@informatik.tu-darmstadt.de

Abstract

Mapping of software onto hardware elements under platform resource constraints is a crucial step in the design of embedded systems. As embedded systems are increasingly integrating both safety-critical and non-safety critical software functionalities onto a shared hardware platform, a dependability driven integration is desirable. Such an integration approach faces new challenges of mapping software components onto shared hardware resources while considering extra-functional (dependability, timing, power consumption, etc.) requirements of the system. Considering dependability and real-time as primary drivers, we present a systematic resource allocation approach for the consolidated mapping of safety critical and non-safety critical applications onto a distributed platform such that their operational delineation is maintained over integration. The objective of our allocation technique is to come up with a feasible solution satisfying multiple concurrent constraints. Ensuring criticality partitioning, avoiding error propagation and reducing interactions across components are addressed in our approach. In order to demonstrate the usefulness and effectiveness of the mapping, the developed approach is applied to an actual automotive system.

1 Introduction

Distributed embedded systems are increasingly defined by software (SW) components[1], determining the functional and *extra-functional* requirements of both safety critical (SC) and non-safety critical (non-SC) operations. Examples of such systems include automotive, avionics and control systems among many others. Implementing these systems using dedicated and partitioned-by-design hardware and separate networks for each component, though desirable for dependability, is prohibitively expensive from the resource, power and space/weight consumption. Consequently, more integrated approaches are often advocated. Integrating SW components of mixed criticality requires careful attention, such that fault-tolerance and real-time requirements are not compromised. Also, new constraints appear as a result of the integration, e.g., limitations in the computing capabilities of a node's processors, restricted memory capacity and bandwidth. Separation between SC and non-SC components is essential and we provide the means to use partitioned hardware nodes to ensure this. *Partitioning* between components consequently prevents error propagation from non-SC to SC components.

A framework for component integration in a distributed embedded systems environment, considering both functional and extra-functional requirements is being developed in DECOS[2] [2]. An important part of component integration is the mapping of components of mixed criticality onto nodes of a shared hardware platform. Extending beyond classical progressive constraint models, we propose a mapping process, that specifically considers both functional and extra-functional constraints of dependability and real-time. The SW mapping is performed to reduce error propagation across hardware (HW) nodes, thus ensuring fault-tolerance over the integration. Rather than focusing on the performance of the algorithm itself, we ensure separation of replicas to maintain dependability over integration, while satisfying timing constraints, minimizing interactions and reducing the communication load on the network.

We consider integration right from the stage where a platform-independent SW specification is available as a basis for the mapping. As shown in [3], the mapping problem is NP hard, which in general requires development of heuristic techniques yielding near-optimal solutions. The mapping problem can be divided into two sub problems: *(i)* assigning different SW components to suitable HW nodes such that platform resource constraints and dependability requirements are met (*resource allocation*) and *(ii)* ordering SW executions in time (*scheduling*).

Our mapping algorithm is a multi phase approach that

[1] A component is a self-contained subsystem that can be used as a building block in the design of a larger system [1].

[2] EC Integrated Project IST DECOS www.decos.at

Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing
0-7695-2561-X/06 $20.00 © 2006 IEEE

explores various combinations of mappings subject to specified constraints. A central part of the algorithm is the use of heuristics for ordering jobs as well as nodes.

The remainder of the paper is organized as follows: An overview of the related work is presented in Section 2. Section 3 introduces the system architecture and models used in the mapping process. The strategies used in the mapping process are described in Section 4. Section 5 provides the details of the process including the allocation algorithm. An example automotive application, with both SC and non-SC components, is used in Section 6 to illustrate the developed approach.

2 Related Work

Many research efforts have addressed the allocation problem in distributed and real-time systems [3, 4, 5, 6, 7]. The generalized resource allocation problem can be modeled as a *constraints satisfaction problem (CSP)*, which is characterized by a set of variables V defined by a corresponding domain D of possible values and a set of constraints C. A solution to the problem is an assignment of a value in D to each variable in V such that all constraints C are satisfied. A number of different techniques have been developed for solving the CSP [8], e.g., constraint propagation [7], inform branch-and-bound and forward checking [5] and backtracking [9]. Techniques used for solving resource allocation problems include mixed integer programming [10] and branch and bound [6]. The major requirements for an embedded real-time system are to meet deadlines and to provide dependability (fault tolerance, avoiding error propagation), but commonly used allocation approaches do not often address these (for example [5, 10]) or address either one or the other. Commonly, when scheduling for real-time systems is performed, a predetermined allocation or a simple allocation scheme is used (for example [11]). If the allocation and scheduling are considered completely separately, important information (e.g., considering constraints) used from one of these activities is missed while performing the other. Existing approaches typically do not address all the constraints or use a limited fault model where dependability is essential. [12] specifically addresses the dependability driven mapping (focuses on minimizing interaction) and presents some heuristics for doing the mapping. However, the focus is on design stage SW objects to aid integration. Unlike other approaches for task/job allocation, the focus of our work is on finding an initial schedulable allocation, a so called off-line allocation of the mixed critical applications (SC and non-SC) onto a distributed platform.

3 System Architecture and Models

Our work considers a time-triggered based distributed architecture [13] that meets the safety requirements of ultra-dependable applications from different domains (e.g., automotive, avionics and control applications). The importance and benefits of SW integration is evident from design concepts in the avionics industry such as the Integrated Modular Avionics [11, 14].

When system functionality from several application domains are considered, it increases design complexity and entails component-oriented system design. The complexity of large systems can only be managed if the overall system can be decomposed into (nearly independent) components with linking interfaces that are precisely specified in both the value and time domain [1]. A nearly independent component of a large distributed real-time system typically performs a specified service (e.g., x-by wire systems).

In the following sections, we discuss the HW and communication model and SW model followed by the fault and constraints model. Partitioning and separation of SC and non-SC components are described in Section 3.2. The interactions between components and between jobs are addressed in Section 3.5.

3.1 The HW and Communication Model

We assume a network topology allowing every HW node to communicate with each other node. A HW node is a self-contained computational element (single- or multiprocessor) connected with a communication controller. A node can also contain additional resources, e.g., sensors, actuators, etc. For achieving strong partitioning between SC and non-SC components, a node can have two physical partitions, each containing a processor, one hosting SC components and the other hosting non-SC components (Option 1). Another possibility is to have a shared processor running

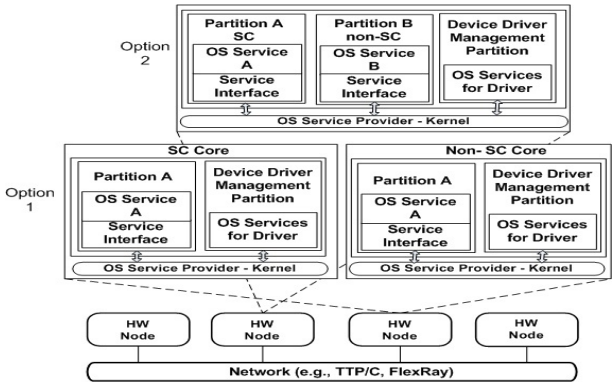

Figure 1. High-level model of the target platform

mixed criticality components (Option 2), as long as strong partitioning is ensured (typically implemented by the OS). Both implementation options for nodes are illustrated in Figure 1 and such platforms are being investigated in the DECOS project. Our approach requires that nodes utilize a

similar configuration, either Option 1 or Option 2. In Figure 1, the OS service provider (kernel) layer is used to virtualize the CPU, dividing it into protected partitions (shown as A, B, etc.), inside which a job executes. The service interface encapsulates the OS services to the specific job running in that partition. The OS kernel layer supports the intra-node's processor communication.

The HW nodes share the same communication channel for inter-node communication to send and receive messages (e.g., by message passing). As our focus is on designing dependable real-time systems, the communication between nodes is based on a fault-tolerant time-triggered network which uses the time division multiple access (TDMA) protocol as its media access scheme. Each node can send messages only during a predetermined time interval, called slot, over a TDMA round. An example of this type of communication is TTP [15], which integrates a set of services (e.g., predictable message transmission, clock synchronization, node membership and redundancy management) necessary for the implementation of dependable real-time systems.

3.2 System Partitioning

Strong partitioning conceptually means that the boundaries among applications (jobs) are well defined and protected so that operations of a job will neither be disrupted nor corrupted by erroneous behavior of another job [16]. Partitioning is needed to ensure that SC components are not affected by the erroneous behavior of non-SC components. Each job is allocated to a single partition as shown in Figure 1, providing a certain amount of computation and memory resources and means to access devices. We allow two ways for ensuring partitioning between SC and non-SC components. Either, *(a)* each node has different processors/cores for the execution of jobs from SC and non-SC components, or *(b)* jobs from SC and non-SC components are combined on the same processor/core in the nodes. Approach *(a)* has the advantage of providing strong partitioning between SC and non-SC components and is less reliant on the use of OS and other partitioning mechanisms. It obviously restricts interactions across SC and non-SC components. This approach can also be beneficial for certification reasons, since partitioning is more obvious. The disadvantage of this approach is that it becomes costly when the number of non-SC applications are high (e.g, continuously increasing comfort/entertainment functionalities in a car), requiring increasing numbers of non-SC components to be considered for the mapping. In this scenario, approach *(b)* works better. Independent of the selected approach, there is a need for partitioning mechanisms [16] in each processor/core which restrict spatial interactions (e.g., sharing error prone resources) and temporal interactions (e.g., one job stealing processor time from another job, or causing a delay of execution of the later) across jobs.

3.3 The SW Model

We use a SW model consisting of components of varied criticality. Components are subsequently decomposed into smaller units, called jobs (Figure 2). A *job* represents the smallest executable SW fragment with basic communication capabilities for exchanging information with other jobs. No knowledge of the internal structure of a job is assumed, allowing for the integration of jobs for which no source code is available.

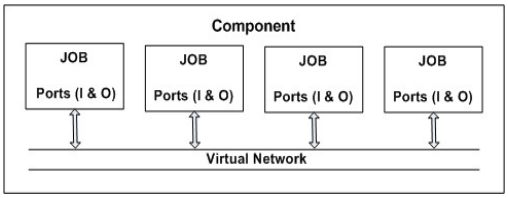

Figure 2. A distributed component

As seen in Figure 2, jobs communicate via input and output ports through a *virtual network*. A virtual network is an overlay network on top of the time-triggered network which possesses a dedicated slot in the underlying TDMA scheme. This way, a dedicated and protected virtual communication channel with known temporal properties is provided to the jobs of a component [17]. We consider a job to have the following properties, which are required as input to the mapping process:

- Job name (each job has a unique name).
- Input and output ports (a job employs input ports for using the services of other jobs, while output ports enable a job to provide services).
- Timing requirements (earliest start time-EST, computation time-CT, deadline-D).
- Inter-node bandwidth requirements.
- Dependability requirements, i.e., degree of replication needed to provide the required level of fault tolerance.

3.4 The Fault Model

We consider both SW and HW faults. For SW faults, each job is considered as a unit for SW error containment. A SW *Error Containment Module* (ECM) [12], should ideally by design, also ensure that errors do not propagate but are contained and tolerated. In case of SW faults, jobs are assumed to fail independently. However, replicas of the same job are not assumed to fail independently since they are based on the same program and use the same input data. Thus, replicated jobs disseminated onto different HW nodes are considered as a single SW ECM. Fault occurrence is considered within an ECM and over communication across them. The consequence of a fault is an error which can propagate from the source to a target, by an erroneous message of a faulty job. SW faults occurring in one job prop-

agate to another job if and only if there is communication between them. The failure of a job is defined as the violation of the specification in either the time or the value domain.

For HW faults, a node is referred to as a HW ECM. A HW node contains resources shared across jobs, e.g., processor, memory and power supply. A single fault impacting any of these shared resources is likely to affect several or all of the jobs running on the node. So the shared resources introduce new paths for error propagation. Hence, replicated jobs should be allocated onto different HW nodes in order to ensure the fault-tolerance of application services in case of HW node failures (e.g., transient fault, crash fault [18]). We assume that only a single HW node becomes faulty within a bounded interval of time.

3.5 Interactions Across Jobs

We distinguish two classes of interactions, depending on the level of detail (component and job) at which the system is observed. The interactions at different levels can lead to harmful error propagation from one module (component or job) to another. The sharing of resources by different jobs may lead to error propagation. Thus, to ensure fault containment, we strive to reduce the interactions while performing the mapping (for details see section 4.2). The issue of error containment plays a crucial role since consequences of a particular error must not have an adverse effect on a SC function. If this cannot be ensured, the error needs to be classified as catastrophic [19].

3.6 Functional and Extra-functional Constraints Model

Constraints define the conditions that limit the possible mappings from the dependability, timing, precedence relation or resource perspectives. We define *hard* constraints that need to be satisfied else the solution is not valid. *Soft* constraints are those whose violation makes the solution inefficient but not completely invalid. Unlike single-constraint driven integration, as mentioned in the related work section, our approach is characterized by a concurrent set of constraints, namely:

- **Binding constraints:** Some jobs can only be mapped on a subset of available nodes due to the need of certain resources (e.g., sensors or actuators).
- **Dependability constraints**
  - SC and non-SC partitioning: In order to maintain strong partitioning between components of different criticality.
  - Separation of replicas: Replicated jobs must be in partitions of different HW nodes.
- **Computing constraints**
  - Computational capability: The sum of computation times of all jobs running on the same processor must be less than the computation capability provided by that processor.
  - Memory: The memory usage of jobs cannot exceed the available memory capacity.
- **Communication constraints:** Sufficient bandwidth for communicating jobs on different nodes must be provided by the network.
- **Timing constraints**
  - Precedence relations: When a job depends on the result of another job.
  - Deadlines: Jobs should finish execution before their deadlines.

4 Mapping Strategies

On the basis of the models and constraints presented in Section 3, we now outline the issues and policies that will drive our mapping process. The strategies presented in this section are employed in the allocation algorithm given in section 5.2. We start by discussing the strategies for ensuring fault tolerance, followed by discussions on the desire to reduce interactions and inter job communication. This assists in providing a more dependable system. Discussions on schedulability to ensure the real-time properties and node restrictions follow, and these are vital to ensure a valid mapping.

4.1 Fault Tolerance (F_t)

The fault tolerance strategy ensures the dependability through replication of jobs from SC components. Fault-tolerance can be enhanced by allocating replicas of jobs onto different nodes and either having recovery replicas to take over when a failure is detected, or use voting to mask the failure of a job. As all jobs from a single component may not be equally critical, we do not replicate whole components. Replication of critical jobs makes the system more dependable. However, brute replication may in turn come at the expense of increased cost, power (more resources may be needed to allocate the additional jobs), and schedulability. The degree of replication of jobs is specified based on the level of desired criticality.

4.2 Communication and Interactions (C_i)

The idea is to minimize the interactions between jobs and therefore the communication load on the physical network by allocating jobs with the highest mutual communication to the same node. This also aids reduction of propagation of errors across jobs. In doing this, it is important not to violate fault tolerance, timing and resource constraints. Communication clustering heuristics, which attempts to allocate highly communicating jobs to the same node, thus reducing the overall communication load on physical network, have been addressed in [4]. Between two communicating jobs, there is an interaction that may lead to propagation of errors from one job to the other. When communication between

two jobs is high, the interaction between them is considered high as well. If a job is affected by an error of the node it is running on, it might propagate errors by interacting with jobs on other nodes. These interactions risk the failure of multiple nodes and are undesirable. Three benefits of assigning communicating jobs to the same node are: *(i)* the error propagation probability between HW nodes is reduced, *(ii)* the communication load on the network is reduced (may allow for the use of a slower but cheaper network [7]) and *(iii)* the total execution time (computation time + time to send/receive messages) of a job is reduced since network delays are avoided. Also, when communicating jobs are placed on different nodes, bandwidth constraints will need to be checked.

4.3 Schedulability (S_t)

The timing constraints defined in section 3.6 ensure the real-time properties of the system. During assignment of jobs the timing constraints are checked to ensure that the mapping is schedulable. We assume that jobs may start executing from their earliest start time and must finish before their deadline. If there are any precedence relations between jobs these must be preserved. When assigning jobs to a processor which already hosts one or more jobs, timing constraints are checked to ensure schedulability. The condition for schedulability presented below is necessary, and if preemptions are allowed, the condition is also sufficient. Since the proposed mapping takes place during the early stages of design, we do not enforce any particular scheduling strategy. Furthermore, there is no restriction on choosing either periodic or aperiodic jobs. As the underlying TDMA based time triggered communication base naturally supports periodic jobs, these are utilized in the example. In general, most control applications such as the example automotive area, often use periodic job sets.

For checking timing constraints, we let $\mathcal{J}$ be the set containing all jobs already assigned to that node, as well as the job we are about to assign. If the difference between maximum end time (highest deadline) and minimum earliest start time for any possible combination of jobs in $\mathcal{J}$ is less than the sum of their computation time, then we cannot assign the job to this processor. If the resulting mapping is to be schedulable, the condition shown below must hold for all subsets of jobs to be allocated on the same processor.

$$max_{\forall j \in \mathcal{J}}(D_j) - min_{\forall j \in \mathcal{J}}(EST_j) \geq \sum_{j \in \mathcal{J}} CT_j,$$

where $\mathcal{J}$ is the set of jobs that are being checked.

Considering an example, two jobs $j_3 \equiv \{EST, CT, D\} \equiv \{2, 4, 10\}$ and $j_4 \equiv \{7, 6, 14\}$ are already assigned to a given node. If we try to map another job $j_5 \equiv \{5, 5, 10\}$ to this node, timing constraints are violated due to this job. Scheduling of these jobs will not be possible (see Figure 3) since $((14-2) < (4+6+5))$, thus j_5 must be assigned to a different node. Of course, the total computing time required by the jobs running on a single processor must not exceed the computational capability provided by that processor.

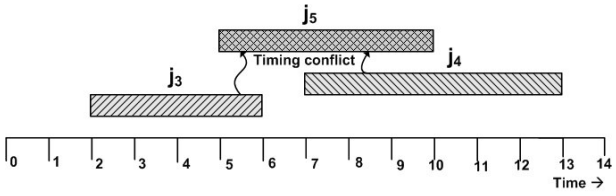

Figure 3. Schedulability analysis

For communicating jobs located on different processors the message transmission time through the network has to be considered to make the jobs schedulable. We assume a maximum network delay T_N, for transmitting a message across the bus. This time must be bounded by using an appropriate protocol, e.g., a statically scheduled TDMA protocol. Of course the network delay depends on whether a node gets access to a TDMA slot for sending messages in this TDMA round or will have to wait for the next round. The deadline of a job sending a message to another job located on a different node must be reduced by the time T_N to accommodate for possible network delays.

4.4 Memory Resources (M_r)

Each job has its own memory requirements. The total amount of memory required by all jobs which are allocated to the same processor, has to be supported by the memory capacity available for jobs on that processor. The utilization of memory available for jobs on a processor should be less than or equal to 100%.

5 The SW-HW Mapping Process

On the background of the constraints (Section 3) and mapping strategies (Section 4), we now present the overall mapping process. The goal of the mapping process is to assign jobs onto available HW resources. In this section we present an approach for the mapping problem based on heuristics. The idea behind the process is to order the jobs and the nodes to facilitate the recursive assignment. Jobs are ordered so that the most conflicting and most constrained jobs are handled first. Similarly, the nodes which allow the most assignments are ordered first. To facilitate strong partitioning between SC and non-SC components, we allow jobs of SC components and jobs of non-SC components to be mapped onto separate processors on the same node.

5.1 Prerequisites of the Mapping Process

The mapping process assumes that a specification of the platform independent SW components and a HW model is available. Properties that need to be satisfied in the mapping are modeled as constraints. All constraints imposed on the jobs of a component are needed before the mapping can take place. Measurements on existing code or estimations can be used to obtain timing information. For SC jobs, the designer has to specify the required degree of replication, to ensure fault-tolerance. Other constraints, such as the computational capability and memory capacity of the computing processors as well as available network bandwidth have to be extracted from the HW model.

5.2 Allocation Algorithm

The allocation is accomplished by using an iterative process. The construction of the algorithm is inspired by the established constructive heuristics in space allocation [20], in course timetabling [21] and by the variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem [22]. The algorithm works in three phases and handles SC components and non-SC components separately to reduce interactions. As a result of component based design, SC and non-SC components communicate minimally, thus they can be treated separately. However, for systems that are not designed using a component based design approach (e.g., when components are reused from a legacy design), this assumption may not be valid. In Phase 1 of the algorithm, we assign replicated jobs of SC components. Next, Phase 2 assigns the non-replicated jobs of SC components. Finally, jobs from non-SC components are assigned in Phase 3.

The described allocation algorithm is executed **once** in **each** phase of the mapping process. We start by considering the most conflicting jobs that cannot be mapped on the same node (i.e, replicas) in the first phase. Throughout the assignment process the most constrained jobs (with respect to binding constraints) are assigned first. Using this ordering of jobs we reduce the necessary backtracking steps. In the second phase, we continue with non-replicated jobs of SC components, they will be integrated with the replicated jobs of SC components, which reduces job interactions. Finally, jobs from non-SC components are allocated in the third phase. A high level description of the work in each allocation phase is outlined in Algorithm 1.

5.3 Heuristics

The above developed algorithm needs two important heuristics to perform well, namely how to decide which job to assign next (see section 5.3.2, ordering of jobs), and which node to assign to this job (see section 5.3.3). This is similar to so called variable (job) and value (node) ordering

Algorithm 1 Allocation Algorithm - executed once for each phase

I: Let J be the set of all jobs to be assigned in this phase.

II: Replicate jobs according to their degree of criticality, so that J now contains a set of replicas. Replicas are represented as $j_{ia\ldots z}$ (e.g., if job j_1 is replicated two times then it is represented as j_{1a}, j_{1b}).
/*Replication only occurs in Phase 1.*/

III: Order the jobs according to a job ordering heuristics, and let J represent the list of ordered jobs.
/*See section 5.3 for a discussion of the heuristics used. */

IV: Order the nodes according to a node ordering heuristics, and let N represent the list of ordered nodes.
/*See section 5.3 for a discussion of the heuristics used. */

V: Select the first job $j_i \in J$.

VI: Select the first node $n_k \in N$ that has not been evaluated already as a possible assignment for j_i.

VII: Evaluate if the selected job j_i can be assigned to node $n_k \in N$. If the assignment is possible, then assign j_i to node n_k, else go back to step VI.
/*See Section 5.3.3 for a detailed discussion*/

VIII: If the assignment was successful proceed to step IX, else a dead-end has been reached(i.e., a valid assignment cannot be found for j_i). When a dead-end is reached then backtrack, i.e., undo one or more previous assignments and try alternative ones (going back to step V). If no feasible solution is found by backtracking, then stop.
/*The backtracking goes back to the next most recently instantiated job, i.e, chronological backtracking.*/

IX: Remove the allocated job j_i from the list J and then repeat (from V) the same procedure until the list J is empty.

heuristics which are concerned with the order in which variables are instantiated and values assigned to each variable. A good variable ordering is one that starts with the variables that are the most difficult to instantiate (i.e., a so-called most constraining variable ordering heuristic) and a good value ordering heuristic is one that leaves open as many options as possible to the remaining uninstantiated variables (i.e., least constraining value ordering heuristic). These heuristics can have a great impact on search efficiency [22]. No backtracking would be necessary if an optimal variable/value ordering is achieved, thus in such a case a linear time solution for the mapping problem is possible. Also, in [8], it is shown that a proper and good selection of variable ordering can reduce the number of steps to find a solution. The following sections describe the data structures used by the heuristics as well as the heuristics used in our mapping process.

5.3.1 Supporting Data Structures

We introduce two matrices for the purpose of ordering jobs and nodes. The assignment matrix A is used to check the usable nodes for each job and accordingly jobs and nodes are ordered. The communication matrix C represents the communication between jobs and can be used to determine the most communicating jobs.

Assignment Matrix A: A rectangular matrix $A_{m \times n}$ is used to describe possible assignments of jobs onto nodes, in such a way that rows represent nodes and columns represent jobs, where m is the total number of nodes and n is the total number of jobs. Note that all replicas of the same job are represented using only one column. Each element of the matrix is filled with either 0 or 1, 1 if a job j_i can be assigned to a node n_k and 0 if it can not. Restrictions on which nodes a job can be assigned to is the result of binding constraints.

Communication Matrix C: A communication matrix of size $n \times n$ is used in order to determine the most communicating jobs. Each element of the matrix corresponds to the communication of a pair of jobs, and n being the number of jobs (counting replicas of the same job only once). If there is communication between the two jobs i and j, we use the value $C_{i,j}$ to represent the total amount of data (bytes) being transferred. If there is no communication, 0 is used. This means that the communication matrix by construction will be symmetric. Note that $C_{i,j}$ denotes the maximum amount of communication possible between jobs i and j (i.e., the available size of sent and received messages as defined by the user).

5.3.2 Job Ordering Heuristics

These heuristics are used to order the jobs, i.e., decide which jobs to assign first. The assignment matrix A and the communication matrix C is used in the ordering as follows:

1a. Create a sub-matrix $\widetilde{A}$ of the assignment matrix A, containing only those jobs (columns) to be assigned in this phase of the allocation.

b. Sum each column (representing a job) in the matrix $\widetilde{A}$. Order the jobs in ascending order, i.e., the jobs with the least possible assignments will come first. Ties are broken according to the second heuristic given below.

2a. For allocation Phases 1 and 3, create a sub-matrix $\widetilde{C}$ of the communication matrix C, containing only those rows and columns belonging to jobs that are to be assigned in this specific phase. For Phase 2, we create a sub-matrix $\widetilde{C}$ of the communication matrix C, containing both the rows and columns belonging to jobs that are to be assigned in this phase, as well as those rows and columns belonging to jobs assigned in Phase 1. The reason for including already assigned jobs in the matrix $\widetilde{C}$ in Phase 2, is that jobs to be assigned in this phase belong to SC components and thus are more likely have communication with the jobs previously assigned in Phase 1.

b. Search the matrix $\widetilde{C}$ and find the pair of jobs with the highest mutual communication between them. Arbitrarily, select one of the jobs in the pair and order that job first, followed by the second job in the pair. If any (or both) of the jobs in the pair have already been ordered, just ignore it. Continue with selecting the next most communicating pair and order those jobs as described, until there are no jobs left. Ties are broken arbitrarily. This heuristic can be applied stand-alone when jobs are not restricted by binding constraints.

Note that for implementing these heuristics, it is not necessary to create the full assignment matrix A nor the full communication matrix C, the sub-matrices will suffice. Further the sub-matrix of the communication matrix, which is used in Phase 1 is itself a part of the sub-matrix used in Phase 2. Hence, the sub-matrix of Phase 2 could be created and used in Phase 1, reducing the number of matrices that need to be created. Also, the symmetry of the communication matrix (and its sub-matrices) can possibly be exploited in the implementation.

5.3.3 Node Ordering Heuristics and Assignment Evaluation

Just as in the job ordering heuristics, we use the same sub-matrix $\widetilde{A}$ of the assignment matrix A for ordering nodes. Nodes are ordered by taking the sum of each row (representing nodes) in the sub-matrix $\widetilde{A}$, and ordering the nodes in descending order. By using this ordering the nodes which allow the most assignments are ordered first. Ties are broken arbitrarily.

Before a job can be assigned to a node, an evaluation has to be performed. If the node is empty, i.e., there are no previously assigned jobs to that node, then only binding constraints and computing node constraints need to be checked. If all nodes provide the resources to handle any

job then checking computing node constraints can be omitted. If there are already assigned jobs on a node then dependability constraints (F_t) such as separation of replicas as well as scheduling requirements (S_t) have to be considered. If a feasible assignment is found then it is used, otherwise exploration continues with the next node. This evaluation ensures that constraints are met.

In the case when non-SC components share the same processor as SC components, some optional strategies are possible. After the jobs belonging to SC components have been assigned, nodes can be re-ordered in a way that eases the assigning of non-SC jobs. As an example, by ordering the jobs according to the amount of remaining computation capacity of each node, we can achieve more load balancing between nodes. Another possibility is to re-order the nodes according to least memory utilization. Both of these re-orderings might also be beneficial from a dependability viewpoint. Since non-SC jobs will be assigned primarily to nodes with few/no jobs from SC components, the separation of SC and non-SC jobs is likely to increase. This will reduce the likelihood of errors occurring in non-SC components propagating to SC components.

6 Mapping Illustration

To illustrate the mapping process of Section 5, we present an example automotive application, consisting of one SC component and one non-SC component. To the best of our knowledge, no mapping processes consider architectures making use of separate processors for SC and non-SC components. However, this type of architecture might be required for implementing highly dependable systems. To show the uniqueness of our approach, in this illustration, we consider such an architecture. We start by describing the HW platform, onto which the jobs are mapped. Next, we describe the components forming our example and the jobs they consist of. We continue by showing how the jobs of the components are mapped onto nodes in the platform during the different phases of the mapping algorithm. Finally, we summarize the results of the mapping.

6.1 HW Platform

The available hardware platform consists of four nodes each containing two processor cores, one for SC components and one for non-SC components. The nodes are connected to a physical network. All nodes have identical processors and the same amount of memory and bandwidth. The memory capacity of each processor of a node is $15kb$ and nodes are connected to a communication channel with a bandwidth of $10kb/s$. Node n_3 has a temperature sensor attached to it.

6.2 SW Components

A brake-by-wire [23] system is used to represent a typical SC component. The non-SC component is a door control system extracted from an actual FIAT vehicle specification. Values for the different properties of the SW components have been chosen to illustrate the mapping algorithm.

6.2.1 Safety Critical Component

The brake-by-wire system in a car is a SC component that must provide service, without endangering human life, in case a fault occurs. The considered brake-by-wire manager system consists of six jobs, namely the brake pedal sensor (BPS) which produces the pedal signal. The brake force control (BFC) uses the pedal signal to calculate the brake forces for the four actuators (BAFR-brake actuator front right, BAFL-brake actuator front left, BARR-brake actuator rear right, BARL-brake actuator rear left). BPS and BFC are safety critical jobs and are replicated three times to run in a TMR setting. A structural model of the brake-by-wire component is shown in Figure 4(a). For simplicity, only one brake actuator (BA) is shown in the figure. The values of

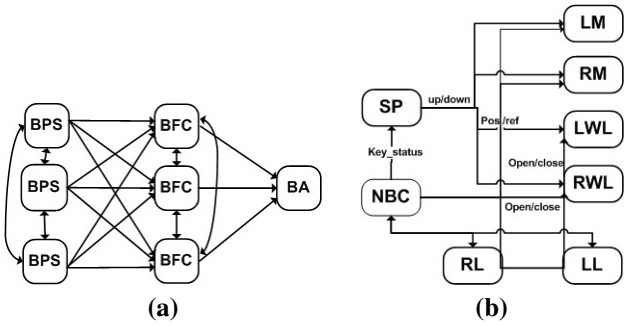

Figure 4. Structural description of the brake-by-wire component (a) and doors component (b).

job properties are shown in Table 1. In the communication column the entry $50 \rightarrow j_2$ indicates that the corresponding row's job j_1 sends 50 bytes of data to job j_2.

6.2.2 Non-Safety Critical Component

The doors control component (assuming a two door car) controls the closing of the doors and windows as well as the heating of the mirrors. We decompose this component into the following 8 jobs. A switch panel (SP) detects the pressing or release of the switches and sends the corresponding commands to the window lifter and mirror. The mirror handler consists of two jobs - one for the left mirror and one for the right mirror (LM and RM) which are designed for moving, heating and comfort closing of mirrors. The window

Job list	Description / name	EST (unit time)	CT (unit time)	D (unit time)	Memory (kb)	Communication (bytes)
j_{1a}	BPS	3	5	14	5	$50 \rightarrow j_2$
j_{1b}	BPS	3	5	14	5	$50 \rightarrow j_2$
j_{1c}	BPS	3	5	14	5	$50 \rightarrow j_2$
j_{2a}	BFC	2	4	12	3	$60 \rightarrow j_3, 60 \rightarrow j_4, 80 \rightarrow j_5, 80 \rightarrow j_6$
j_{2b}	BFC	2	4	12	3	$60 \rightarrow j_3, 60 \rightarrow j_4, 80 \rightarrow j_5, 80 \rightarrow j_6$
j_{2c}	BFC	2	4	12	3	$60 \rightarrow j_3, 60 \rightarrow j_4, 80 \rightarrow j_5, 80 \rightarrow j_6$
j_3	BAFR	2	7	17	4	
j_4	BAFL	4	6	14	4	
j_5	BARR	7	6	20	6	
j_6	BARL	5	7	18	6	

Table 1. Chosen values of job properties (brake-by-wire manager)

Job list	Description / name	Binding functionality	EST (unit time)	CT (unit time)	D (unit time)	Memory (kb)	Communication (bytes)
j_7	NBC		3	7	15	7	$40 \rightarrow j_8, 30 \rightarrow j_{13}, j_{14}, 30 \rightarrow j_9, j_{10}, j_{11}, j_{12}$
j_8	SP		3	6	16	8	$40 \rightarrow j_9, j_{10}, j_{11}, j_{12}$
j_9	LM	Temp Sensor	4	4	13	3	
j_{10}	RM	Temp Sensor	4	4	12	3	
j_{11}	LWL		3	8	17	6	
j_{12}	RWL		3	8	17	6	
j_{13}	RL		5	5	20	4	$30 \rightarrow j_7$
j_{14}	LL		5	5	20	4	$30 \rightarrow j_7$

Table 2. Chosen values of job properties (doors component)

lifter consists of two jobs for lowering and raising the windows, left front window lifter (LWL) and right front window lifter (RWL). The lock-unlock function is responsible for locking, unlocking and dead-lock functionality of the door. It comprises two jobs: left lock-unlock (LL) and right lock-unlock (RL). The body computer node (NBC) coordinates the operation of the other jobs (mirror, window lifter and lock-unlock) and sends the status information to the switch panel. A structural model of the doors component is shown in Figure 4(b). The values of job properties are outlined in Table 2.

6.3 Mapping Phases

We proceed by describing the allocation of jobs that takes place during the three phases of our allocation algorithm depicted in Section 5.2.

6.3.1 Phase 1: Assignment of Replicated Jobs of SC Components

In Phase 1 we consider the high-criticality jobs of SC components. In this example the high-criticality jobs are j_1 and j_2, which both have a degree of criticality equal to three. We replicate the jobs to get the set of jobs to be assigned in this phase. This gives us the following jobs to consider: $j_{1a}, j_{1b}, j_{1c}, j_{2a}, j_{2b}, j_{2c}$. We then proceed by creating the allocation matrix A and the communication matrix C, which will assist in the ordering of jobs and nodes. As none of the jobs require any special features, they can both be assigned to any node. Hence, the sub-matrix $\widetilde{A}$ relevant for Phase 1 will contain only ones. Using the first job ordering heuristic results in a tie. Thus, the jobs are ordered using the second heuristic. Since, we are only assigning two jobs (not considering replicas). Using the communication heuristics also results in a tie. Thus the jobs are ordered arbitrarily, lets assume they are ordered as $j_{1a}, j_{1b}, j_{1c}, j_{2a}, j_{2b}, j_{2c}$.

Now, we try to order the nodes using the sub-matrix $\widetilde{A}$. The result is a tie, since all jobs can be assigned to all nodes. This means that the nodes can be ordered arbitrarily, lets say that they are ordered as n_1, n_2, n_3, n_4. We then select job j_{1a} and successfully try to assign it to node n_1. Then we select job j_{1b} and try to assign this to node n_1 as well. However, the node assignment evaluation (step VII) shows that this is impossible since it violates the requirement that replicas should be separated (F_t). Consequently, job j_{1b} will be assigned to the next node n_2 and job j_{1c} will be assigned to node n_3. Similarly, the three replicated jobs of j_2 will be assigned to the first three nodes. The node assignment evaluations show that these assignments can be performed without any violation of constraints. This concludes Phase 1.

6.3.2 Phase 2: Assignment of non Replicated Jobs of SC Components

In Phase 2, we try to assign the non-replicated jobs of SC components. In the example, this means jobs j_3, j_4, j_5, j_6. Again, the assignment matrix A cannot be used for deciding the job ordering. Thus we use a sub-matrix $\widetilde{C}$ of the communication matrix C, to decide the job ordering. This reduces interactions and communication load on the network. The matrix used is shown in Table 3. As discussed in the job ordering heuristics section, this sub-matrix also contains the already assigned jobs j_1 and j_2. Applying the communication heuristics we come up with the following job ordering: j_5, j_6, j_3, j_4 (with ties broken according to lowest index first).

Trying to order the nodes results in a tie and lets (again) assume the ordering n_1, n_2, n_3, n_4. Job j_5 can be assigned to node n_1 without violating any constraints. Job j_6 cannot be assigned to n_1 due to a timing constraint violation. The highest/maximum deadline among those jobs assigned (j_{1a}, j_{2a}, j_5) and the job about to be assigned (j_6) to n_1 is 20 and the lowest earliest start time is 2. Thus, the difference is 18, while the sum of computation times is $(5 + 4 + 6 + 7) = 22$, which is greater than 18. Consequently, job j_6 is assigned to the next explored node, i.e., to n_2. Since there is not enough memory capacity, job j_3 cannot be assigned to either of nodes n_1 or n_2. Also, assigning j_3 to any of the nodes containing replicas of j_1 and j_2 would violate timing constraints. Therefore, job j_3 will be assigned to node n_4. The same reasoning holds for j_4, which is also assigned to the node n_4. The allocation resulting from the execution of Phase 1 and Phase 2 for the SC component is shown in Table 4.

Job list	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	50	0	0	0	0
j_2	50	0	60	60	80	80
j_3	0	60	0	0	0	0
j_4	0	60	0	0	0	0
j_5	0	80	0	0	0	0
j_6	0	80	0	0	0	0

Table 3. The sub-matrix $\widetilde{C}$ used in Phase 2

6.3.3 Phase 3: Assignment of Jobs of non-SC Components

In this phase we consider the jobs of the non-SC doors component. Jobs are ordered according to the sub-matrix $\widetilde{A}$ of the assignment matrix A. As jobs j_9 and j_{10} need temperature sensors, the only node usable for them is node n_3. Since these jobs have the least number of usable nodes, they are ordered first. All other jobs have the same number of usable nodes. The communication heuristics is used to order all tied jobs. Jobs j_9 and j_{10} are still tied for first place and the job ordering becomes as follows: $j_9, j_{10}, j_7, j_{13}, j_{14}, j_8, j_{11}, j_{12}$ (with remaining ties broken by lowest index first).

Nodes are then ordered using the sub-matrix $\widetilde{A}$. Node n_3 is ordered first, since it is the least constrained node, allowing all jobs to be mapped onto it. The rest of the nodes are tied and we consider the following order in the example: n_3, n_1, n_2, n_4 (with ties broken according to lower index first).

Jobs j_9 and j_{10} are mapped onto node n_3. This is the only feasible mapping, since this node has the required temperature sensor. During the assignment evaluation, both schedulability (S_t) and resource constraints are checked. Job j_7 is selected to be assigned next. It cannot be assigned to node n_3 since this would violate schedulability (S_t). Consequently it is mapped onto the next node n_1. Jobs j_{13} and j_{14} will also be mapped onto node n_1 for the same reason. Due to violation of constraints (timing and memory), jobs j_8 and j_{11} are assigned to node n_2. Finally, job j_{12} is assigned to node n_4 since no other assignment is possible. The resulting allocation is shown in Table 4.

6.4 Comments

The resulting allocation is shown in Table 4. As can be seen, the first node runs three jobs from the SC component and three jobs from the non-SC component. The second node runs three jobs from the SC and two jobs from the non-SC. The third node runs two from the SC and two from the non-SC and the fourth node runs two jobs from SC and one job from the non-SC component.

	n_1	n_2	n_3	n_4
SC core	j_{1a}, j_{2a}, j_5	j_{1b}, j_{2b}, j_6	j_{1c}, j_{2c}	j_3, j_4
non-SC core	j_7, j_{13}, j_{14}	j_8, j_{11}	j_9, j_{10}	j_{12}

Table 4. Resulting allocation of jobs from the brake-by-wire and doors components

7 Conclusion and Future Work

We have presented a *dependability* driven mapping approach for integration of different criticality components onto a shared distributed platform. The mapping strategies and algorithm presented in this paper has features such as effectiveness (*quality of the solution*), efficiency (*reducing the search space and finding a feasible solution*) and robustness (*consistent to perform the same mapping over many runs*). The algorithm uses comprehensive strategies, with the formulation of the constraints, when integrating SC and non-SC components onto the same node unlike existing approaches. The job and node ordering heuristics allowed us to apply look-ahead analysis that is to decide which job to

allocate next (*job ordering*) and which nodes to assign to each job (*node ordering*). Allocating jobs according to our communication heuristics reduces interactions and the communication load on the network. Our developed approach considered dependability issues in three aspects: providing fault tolerance, minimizing interactions and allowing partitions for the desired system design. The schedulability test during job assignment ensures that the output of the allocation can be scheduled.

Our future work includes implementation of the algorithm by using an optimization scheduling tool and implementation in a real HW platform by utilizing the examples presented in this paper. The developed approach is being integrated into a tool suite where the intention is to convert initial platform independent SW components into a platform specific model post integration. We are also looking for a near-optimal solution by applying multi variable optimization techniques, such that multiple objectives are addressed and the optimality of a solution is evaluated.

Also, we plan to extend the scheduling analysis part and consider including communication scheduling in the approach. Further, it would be interesting to compare the effect of using different heuristics and the impact they have both on the final solution, but also the time it takes for the algorithm to execute.

Acknowledgments

This work has been partly supported by the European IST project DECOS (EC IST FP6 IP DECOS) No. 511764. We would like to thank all members of the DEEDS group for the valuable discussions and comments on the earlier version of the paper.

References

[1] Kopetz, H. et al. *Compositional Design of RT Systems: A Conceptual Basis for Specification of Linking Interfaces*. In Proceedings of the 6th IEEE ISORC, pp. 51–60. May 2003.

[2] DECOS. *Dependable Embedded Components and Systems*. IST, EU FP 6, http://www.decos.at.

[3] Fernandez-Baca, D. *Allocating Modules to Processors in a Distributed System*. IEEE Trans. on Soft. Eng., 15(11): pp. 1427–1436, 1989.

[4] Ramamritham, K. *Allocation and Scheduling of Precedence-Related Periodic Tasks*. IEEE Trans. Parallel Distrib. Syst., 6(4): pp. 412–420, 1995.

[5] Wang, S. et al. *Component Allocation with Multiple Resource Constraints for Large Embedded Real-Time Software Design*. In IEEE RTAS, pp. 219–226. 2004.

[6] Kuchcinski, K. *Constraints-driven Scheduling and Resource Assignment*. ACM Trans. Design Autom. Electr. Syst., 8(3): pp. 355–383, 2003.

[7] Ekelin, C. et al. *Evaluation of Search Heuristics for Embedded System Scheduling Problems*. In Proceedings of the 7th Int. Conf., Constraint Programming, pp. 640–654. 2001.

[8] Kumar, V. *Algorithms for Constraint-Satisfaction Problems: A Survey*. AI Magazine, 13(1): pp. 32–44, 1992.

[9] Dakic, T. et al. *Backtrack-Free Search for Resource Allocation Problems*. Technical Report, CMPT TR 95-2, School of Computing Science, Simon Fraser University, Burnaby, B.C, CANADA V5A 1S6, January 30 1995.

[10] Rajkumar, R. et al. *Practical Solutions for QoS-Based Resource Allocation*. In RTSS, pp. 296–306. 1998.

[11] Lee, Y. et al. *Resource Scheduling in Dependable Integrated Modular Avionics*. In Proc. of the Int. Conf. on Dependable Systems and Networks, pp. 14–23. 2000.

[12] Suri, N. et al. *A Framework for Dependability Driven Software Integration*. In Proceedings of the 18th Int. Conf. on Distributed Computing Systems, ICDCS, p. 406. 1998.

[13] Kopetz, H. et al. *The Time-Triggered Architecture*. Proceedings of the IEEE, 91(1): pp. 112 – 126, Jan. 2003.

[14] ARINC. *Design Guidance for Integrated Modular Avionics*. Arinc report 651, Aeronautical Radio Inc., Annapolis, MD, Nov 1991.

[15] Kopetz, H. et al. *TTP-A Protocol for Fault-Tolerant Real-Time Systems*. Computer, 27(1): pp. 14–23, 1994.

[16] Rushby, J. *Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance*. NASA/CR-1999-209347, SRI International, Menlo Park, California, 1999.

[17] Kopetz, H. et al. *From a Federated to an Integrated Architecture for Dependable Real-Time Embedded Systems*. Tech. rep. 22, TU Vienna, July 2004.

[18] Laprie, J.-C. et al. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Trans. Dependable Secur. Comput., 1(1): pp. 11–33, 2004.

[19] Kopetz, H. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[20] Silva, J. L. *Metaheuristic and Multiobjective Approaches for Space Allocation*. Phd thesis, University of Nottingham, UK, November 2003.

[21] Rossi-Doria, O. et al. *An Hyperheuristic Approach to Course Timetabling Problem Using an Evolutionary Algorithm*. Tech. rep., School of Computing, Napier University.

[22] Sadeh, N. et al. *Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem*. Artificial Intelligence, 86(1): pp. 1–41, 1996.

[23] Hedenetz, B. et al. *"Brake by Wire" without Mechanical Backup by Using a TTP Communication Network*. In SAE World Congress, pp. 296–306. 1998.