Design Automation for Embedded Systems manuscript No.
(will be inserted by the editor)

An Optimization Based Design for Integrated Dependable Real-Time Embedded Systems

Shariful Islam, Neeraj Suri
András Balogh, György Csertán, András Pataricza

Received: date / Accepted: date

Abstract Moving from the traditional federated design paradigm, integration of mixed-criticality software components onto common computing platforms is increasingly being adopted by automotive, avionics and the control industry. This method faces new challenges such as the integration of varied functionalities (dependability, responsiveness, power consumption, etc.) under platform resource constraints and the prevention of error propagation. Based on model driven architecture and platform based design's principles, we present a systematic mapping process for such integration adhering a transformation based design methodology. Our aim is to convert/transform initial platform independent application specifications into post integration platform specific models. In this paper, a heuristic based resource allocation approach is depicted for the consolidated mapping of safety critical and non-safety critical applications onto a common computing platform meeting particularly dependability/fault-tolerance and real-time requirements. We develop a supporting tool suite for the proposed framework, where VIATRA (VIsual Automated model TRAnsformations) is used as a transformation tool at different design steps. We validate the process and provide experimental results to show the effectiveness, performance and robustness of the approach.

Keywords Fault-Tolerance, Real-Time, Constraints, Mapping, Transformation.

This work has been partly supported by the EU IST FP6 DECOS.

S. Islam, N. Suri
Department of Computer Science, Technische Universität Darmstadt
Hochschulstr. 10, 64289 Darmstadt, Germany. Tel.:(+49) 6151 16 7066, Fax: (+49) 6151 16 4310
E-mail: {ripon,suri}@cs.tu-darmstadt.de

A. Balogh, Gy. Csertán,
OptXware Research and Development Ltd.
Budafoki út 187-189. A. 2. em 32., H-1117 Budapest, Hungary. Tel: (+36) 1 814 9056, Fax: (+36) 1 814 9057
E-mail: {balogh,csertan}@optxware.hu

A. Pataricza
Department of Measurement and Information Systems, Budapest University of Technology and Economics
Magyart Tudósok körútja 2., H-1117 Budapest, Hungary. Tel.:(+36) 1 463 3595, Fax:(+36) 1 463 2667
E-mail: pataric@mit.bme.hu

1 Introduction

Design of dependable[1] real-time (RT) embedded systems comprises diverse functional and critical applications and faces a wide range of competing constraints (e.g., cost, space, weight, power, FT, hard RT and multiple other realization constraints) imposed by the increasing number of applications and their software (SW). Examples of such systems include automotive, avionics and control systems among many others. As these are often safety-critical environments, the applications (*SW component*) are desired to produce correct output and preserve the safety of the operations even in the presence of some faults from an anticipated set of faults. Moreover they have to fulfill several *responsiveness* and *performance* requirements in addition to functional correctness making the design even more challenging [1]. Thus efficient and cost effective system design strategies are needed to integrate these diverse critical applications across limited hardware (HW) resources while considering the interplay of dependability/FT and RT requirements.

Traditional design techniques such as federated approach [2] are increasingly limited for developing such systems. *Extra-functional* properties such as timeliness, FT and safety are introduced often late in the development process when the design is difficult and costly to change/upgrade. For example, FT is treated as an add-on requirement in the design process. A typical (and costly) approach being replicating the implementation, i.e., a so called federated approach. Investigations show that this approach fails to produce cost-effective dependable systems [3]. On the other hand embedded products have become increasingly complex and must be developed quickly that current design methodologies are no longer efficient [4]. Therefore integrated approaches are more often advocated where integration of different criticality applications onto a common computing platform is needed. The importance and benefits of such approach is evident from the design concept in avionics industry such as the Integrated Modular Avionics (IMA) [5–7] as well as such design concept is currently being introduced in automotive industry such as in AUTomotive Open System ARchitecture (AUTOSAR) [8] and in [9, 10]. In order to ease the design complexity, integrated system design should come up with guidelines, methodologies and tools [11] and need a stepwise design process. This method for designing embedded systems requires to specify and design SW and HW separately. The development of such systems also calls for new forms of abstraction and design methodologies for bridging applications with platform details.

The emerging Model Driven Architecture (MDA) [12] and Platform Based Design (PBD) [4] initiatives address such design processes at different abstraction levels. Adhering to these methodologies and for developing an efficient transformation based system design, we propose the following guidelines: *(i)* to start the design by representing the functional application development in an abstract form independent from the platform implementation details, *(ii)* the selection of the HW platform such that it can support the functionality while meeting the performance and dependability/FT requirements, and *(iii)* the *integration/mapping* of application functionalities onto available platform resources satisfying specified design *constraints*. In PBD technology this is often termed as *meeting-in-the-middle* process. This design step faces new challenges under platform resource constraints and needs careful attention such that FT and RT requirements are not compromised. In order to tackle all these design considerations, new methodologies need to be developed. Moreover a suitable toolchain is essential for such design steps in order to be able to design the system in an efficient and cost effective way. Consequently, in this work we propose a formalization of the software job and hardware platform to perform constrained job mapping between them.

[1] The terms dependability and fault-tolerance (FT) will be used synonymously in the paper.

1.1 Our Contributions

Unlike existing traditional design approaches, we consider integration of different criticality applications using a transformation and mapping approach. Using these approaches and adhering to model-based design principles[2], our aim is to convert initial platform independent SW component specifications into a platform specific post integration model (Figure 1). We focus on mapping different applications/SW components onto shared HW resources subject to operational constraints. Applications are further decomposed into smaller executable fragments called jobs. A mapping is defined as: *(I)* assigning jobs onto suitable HW nodes such that platform resource constraints and dependability/FT requirements are met (*resource allocation*) and *(II)* ordering job executions in time (*scheduling*). This particular problem is often NP hard [13] to solve in a tractable manner where a solution can be found in polynomial time [14]. Consequently heuristic solution techniques are often utilized. Also, existing approaches usually do not address *(I)* and *(II)* together. Mostly scheduling is performed assuming a predetermined manual allocation. This may not be possible for a rapidly developed embedded systems where functionalities and complexities (due to large number of design constraints and requirements) are increasing day-by-day. Thus intuitive mapping decisions are inherently limited beyond a given complexity. We have developed a heuristics based *systematic resource allocation* approach for the mapping in [15]. Dependability/FT and RT requirements are the prime drivers for our proposed mapping and both of them are taken into consideration in step *(I)*. The same concept is utilized in this transformation based design process. Rather than focusing solely 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. The output of the algorithm is a feasible mapping of jobs onto HW nodes.

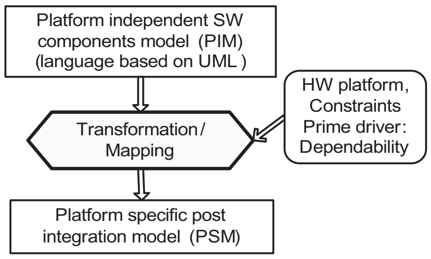

Platform independent SW components model (PIM) (language based on UML)
HW platform, Constraints Prime driver: Dependability
Transformation / Mapping
Platform specific post integration model (PSM)

Fig. 1 Transformation based design

The design starts from the high-level abstraction of system functionality which is completely independent of platform specific programming details. The Platform Independent Model (PIM) is created by specifying the functional as well as responsiveness and dependability properties of application jobs. We develop this model for varied applications (model is given in Section 8.2). The challenging task is to integrate these different criticality PIMs onto resource constrained HW platform. Our aim is to provide an interactive (semi-automated) and iterative transformation based design and a supporting tool set for such design. In the course of the transformation (PIM-to-PSM mapping), applications of different requirements are allocated and integrated onto common HW resources based on the specified constraints. The post integration phase is defined in the Platform Specific Model (PSM),

[2] This follow the pervasiveness of these techniques in industry.

where system functionalities are already mapped onto platform meeting the requirements and *objectives*. This model controls the deployment of executables to the target platform. Over all, we make the following contributions:

1) A novel transformation based design methodology is developed for integrated mapping of SW components onto HW. To the best of our knowledge this is the first transformation based mapping approach that combines both dependability and RT aspects.
2) Relevant design criteria such as classification of requirements and constraints, criticality partitioning, reusability, fault-tolerance, fault-containment, responsiveness, utilization of bandwidth are comprehensively addressed in our approach.
3) We model functional and *extra-functional* requirements in the same abstract platform, i.e., in the PIM. Thus, the extra-functional requirements of RT and dependability/FT are taken into account at early design phase.
4) Dependability is ensured through replication of jobs with high criticality. We then enhance dependability by using fault-containment mechanism and present a schedulability analysis for guaranteeing the responsiveness/timeliness properties.
5) The *marked PIM* is introduced in the design process in order to complement the information of PIM and HW platform model by designer decisions.
6) Based on heuristics a systematic mapping of Safety Critical (SC) and non-SC applications onto a distributed computing platform is carried out such that their operational delineation is maintained over the integration. Our proposed algorithm generates an initial feasible solution and guides the design optimization in an efficient way.
7) We perform extensive experiments which show the effectiveness (quality of the solution), performance (reducing the search space and finding a quick feasible solution) and robustness (consistent to perform the same mapping over many runs) of our design process. A validation of the allocation process is performed as well.
8) We provide a supporting tool set and technologies in a VIATRA[3] based framework. Once all the defined design steps/transformations are done in VIATRA, the PSM of the target system is generated.

1.2 Paper Organization

The paper is organized as follows. Section 2 discusses the related work. Section 3 depicts the fundamental aspects of our target system design describing system requirements, different partitioning policies and SW reusability. The system model and problem formulation is detailed in Section 4. The developed transformation approach is provided in Section 5, where we briefly describe the system design flow. The mapping process is systematically described in Section 6 including mapping strategies (e.g., providing FT, influence/fault-containment, schedulability analysis), proposed heuristics and the algorithm. Section 7 illustrates the mapping using a SC-application from an actual automotive system and provides performance evaluation of the heuristics. The over all process is implemented in a VIATRA based tool suite detailed in Section 8.

2 Related Work

Varied techniques have already been used for solving the resource allocation problem, e.g., constraint propagation [17, 18], inform branch-and-bound and forward checking [18, 19] and mixed integer programming [20]. These approaches typically perform the mapping (allocation and scheduling) straightforwardly applying the above mentioned techniques. A disadvantage of these approaches is that usually they do not put additional efforts to reduce the search space a priori while solving the problem thus limiting their applicability to handle

[3] VIATRA (VIsual Automated model TRAnsformations) is an open source model transformation tool [16].

only a few constraints. [17] applies symmetries exclusion to reduce the search space which is more desirable in a homogeneous system. An enhancement of the Quality-of-Service (QoS) based resource allocation model [20] is presented in [21], where a hierarchical decomposed scheme by dealing with smaller number of resources is described enabling QoS optimization techniques for large problems. Tasks replication is used as a QoS dimension in order to provide FT. In this paper, we decompose our approach into several subproblems and phases in order to reduce the complexity of solving the problem. We describe this more in Section 4.2.

The major requirements for designing embedded systems are to meet both RT requirements and to provide dependability (FT, avoiding error propagation, etc.). Commonly used approaches typically address RT and FT on a discrete basis [19,21]. AIRES (Automatic Integration of Reusable Embedded Systems) [19,22] describes the allocation of SW components onto HW platforms for RT and embedded applications satisfying multiple resource constraints. They also provide a schedulability analysis. The method has been implemented into a Model Driven Development (MDD) analysis tool that evaluates whether those constraints are satisfied. Based on constraint programming, [18] presents an approach to constraint-driven scheduling and resource assignment. They develop a constraint solver engine which satisfies a set of constraints. However dependability/FT is not considered in any of these approaches. Moreover when scheduling for RT systems is performed, a predetermined allocation or a simple allocation scheme is used (e.g., [5]). If the scheduling is performed without assuming any pre-allocation it may significantly increase the computation complexity and can make the problem intractable (cannot be solved in polynomial time [14]). Also if the allocation and scheduling are considered separately, important information (e.g., considering constraints) used from one of these activities is missed while performing the other. On the other hand, usually FT is applied to an existing scheduling principle such as rate-monotonic or static off-line either by using task replication [23] or task re-execution [24]. Existing all these approaches typically do not address all the constraints or use a limited fault model where dependability is essential. We apply *constraints prioritization* [25] during the allocation phase in order to satisfy the constraints which also reduces the complexity to solve the problem. [26] specifically addresses the dependability driven mapping (focuses on minimizing interaction) and presents the heuristics for doing the mapping. However the focus is on design stage SW objects to aid integration. A survey of various SW development processes addressing dependability as extra-functional requirements at both late and early phases is described in [27]. Utilizing model-based principle, [28] describes a component integration method for designing and deploying avionics systems. [29] provides a tool suite for the design and analysis of large-scale embedded RT systems.

Using MDA and PBD methodologies, we develop a rigorous dependable RT embedded system design approach considering all the requirements early in the design process as well as provide a detail description of a heuristics based allocation and scheduling. Furthermore we provide a new supporting tool-chain for the design of both SC and non-SC applications.

3 Preliminaries

This section describes the functional and extra-functional requirements, partitioning issues for SW execution and the SW reusability, which are the core criteria for our integrated system design approach. Based on these descriptions, in Section 4.1 we present the system model comprising of SW and HW model, constraints and the fault model.

3.1 Requirements

A typical FT-RT embedded system has to comply to a set of frequently contradicting requirements formulating both envisaged functional and extra-functional properties. The requirements can be expressed by categorizing them as:

- *constraints*, which have to be satisfied in the target design in a mandatory way. Typical representatives are timeliness constraints in hard RT systems or replication level in FT systems.
- *design objectives*, occasionally referred to as *soft constraints*, can serve as a comparison basis between design alternatives by providing some quantitative characteristics expressing the level of compliance of a candidate design to the requirements.

A typical set of requirements for hard FT-RT embedded systems may cover the following:

Temporal Requirements: A hard RT system must respect all timeliness requirements in order to deliver a predictable and deterministic behavior in addition to the compliance to the functional specification. For instance, applications must terminate their execution within a certain temporal limit even in presence of faults. Classical scheduling theories define two types of temporal constraints: *deadlines* to be kept by the termination of execution of the individual jobs and *precedence relations* require a guaranteed termination of a job prior of launching another one causally depending on its results.

Synchronous system implementation (e.g., time-triggered [30]) is a frequently used paradigm guaranteeing by principle the fulfillment of temporal requirements. Here each operation gets a time slot assigned according to its worst-case (longest) execution time.

Dependability Requirements: This class of requirements may contain any of the aspects of dependability properties [31], which are reliability, availability, safety, security, integrity and maintainability. In the case of replication based safety, the top priority constraint relates the number (or cumulated reliability) of replicas to the designated reliability of the system. The next level constraint formulates the requirement to ensure dependability by design, two types of requirements are defined. *Separation of replicas*: replicated jobs from the high critical applications must be in partitions of different HW nodes and *SC and non-SC partitioning*: in order to maintain strong partitioning between applications of different criticality particularly to ensure that SC applications are not affected by the erroneous behavior of non-SC ones.

Resource Requirements: We define several resource related constraints under this category of requirements mentioned as follows. Some jobs can only be mapped on a subset of available nodes due to the need of certain resources (e.g., sensors or actuators) and treat them as binding requirements. The sum of computation times of all jobs running on the same processor must be less than the computation capability provided by that processor (depends on available processor utilization) and the memory usage of jobs cannot exceed the available memory capacity. Sufficient bandwidth for communicating jobs on different nodes must be provided by the underlying network (e.g., TTP/C [32], FlexRay [33]).

3.2 Robust Partitioning Policies

Conceptual partitioning means that the boundaries among jobs as well as among applications are well defined and protected so that operations of a job will neither be disrupted nor corrupted by the erroneous behavior of another job [2]. This erroneous behavior of a job can be the result of a SW fault or a failure in a HW element used exclusively by that job. Partitioning is needed to ensure that SC applications are not affected by the erroneous behavior of non-SC applications. The main means of achieving robust partitioning is the implementation of well-defined and protected damage confinement regions between components assuring a guaranteed blocking of inter-component error propagation. The different policies can be distinguished according to the granularity of the architecture, i.e., the notion of components they apply:

– **Node-level partitioning** is a traditional policy adopting the granularity of HW nodes as elementary construction and fault isolation components. Each (usually highly dedicated) HW node runs a single functional component of the system. Similarly, replication is introduced at the HW node level. The same separation principle is used for isolating the implementations of replicas and SC and non-SC functionalities, as well, by strictly deploying each function onto a separate HW node. Damage confinement isolates faulty components.
  This paradigm necessitates a high HW overhead due to the redundancy induced by the architectural granularization for fault isolation. It typically results in an architecture composed of at least one separate computing node per each individual function interconnected by a fabric of point-to-point communication links.

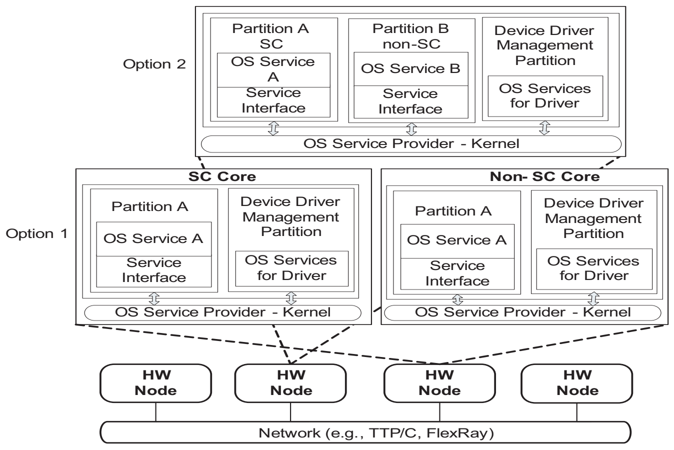


Fig. 2 High-level model of the target platform shows the partitioning concepts and the application execution environment

– **Processor level physical partitioning** is a modified version of node-level partitioning for nodes embedding multiprocessors for a higher computational power. The loose coupling between processors provides a basic inter-processor isolation to take processors as partitioning units, while sharing the remaining (less critical) shared resources such as external communication channels or other I/O interfaces. This is shown in Option 1 in Figure 2. This configuration gives the provision of assigning SC and non-SC applications onto different processor cores on the same HW node so that the influence[4] from non-SC to SC applications are prohibited by design.
  However the feasibility of the principle of system composition of dedicated parts is limited in scope for ever increasing function complexity of embedded systems. A large number of HW components interconnected by a complex fabric becomes prohibitively

[4] Influence is the probability of error propagation between modules where a module can be an application, a job, a processor core or a node.

expensive from the point of view of resource use, power consumption and space/weight. Moreover the high level of redundancy rapidly results in reduced overall system reliability despite the increase in component reliability as induced by technology development. This paradigm necessitates the use of a high-level of HW redundancy. The architectural granularization results in architectures composed of at least one separate computing node per function interconnected by a fabric of point-to-point communication links.

– **Job level partitioning policies** use a finer granular approach by taking jobs as allocation and replication units. It allows HW resource sharing by the deployment of multiple functions (implemented as isolated SW jobs) onto the same HW component (option 2). They provide each job with a certain amount of computational time and memory resources for an exclusive use called the *partition* for the job (Figure 2). Typically, bus organized communication channels interconnect the HW nodes. Safety of the systems is still based on replication of the critical parts. The operating system (OS) service provider (kernel) layer (Figure 2), is used to virtualize the CPU, dividing it into protected partitions (shown as A, B, etc.), inside which a job executes. In a non-SC partition more than one job can run. The service interface encapsulates the OS services to the specific job running in that partition. The OS kernel layer supports the intra-nodes processor communication.

Resource sharing is obviously beneficial from the view point of cost reduction, but inter-job isolation becomes crucial for safety. Partitioning mechanisms [2] in each shared processor have to exclude both erroneous spatial interactions (e.g., *error propagation* via shared resources between jobs) and temporal ones (e.g., starvation of a job caused by another one stealing its processor time). Isolation of the jobs is carried out by means of the standard support mechanism built-in into most modern processors, like memory segmentation in the Memory Management Unit (MMU) further enforced by specialized HW and OS, like [2]. Strict spatial and temporal isolation is provided in platforms intended for SC applications (like TTP) by means of extra HW units assuring FT for each main isolation-related functionality both in the multitasking run-time environment and in the communication infrastructure over shared buses. However in this paper we provide the notion of reducing error propagation during integration so that the partitioning will be less reliant on the use of OS and other partitioning mechanisms.

3.3 Component Reusability

Reuse of existing components is a key approach aimed at reducing development and manufacturing times and costs. An efficient workflow covering all phases of the development process (design, component integration, validation and testing, certification) is one of the key factors in the reduction of development and manufacturing costs and time.

– As job level partitioning loses the dependence on the level of dedication of the HW platforms, they may increasingly become of a generic type, thus supporting the reuse of COTS and other legacy components. Automotive, avionics, control and seaborne systems are representative examples of SC systems relying on a rapidly growing number of *SW components* and a *HW component integration* system design paradigm.
– Another evolving form of reuse is that of the *intellectual property*. While the change in the functionality offered by subsequently developed members of a product family follows typically an evolutionary path, their implementations can drastically differ due to the revolutionary changes in the HW platform technology background.

Note, that there is an interesting interplay between reusability and robust partitioning in building SC systems [6]. As the core concept of robust partitioning is a strict componentization assigning a single partition to each individual functionality to be executed, modification of a functional component influences only those ones, which are in an explicit functional interdependence with it. As side effect freedom is guaranteed by principle with respect to other ones, robust partitioning facilitates the reuse, modification, debugging, integration, and certification of components.

4 System Model and Problem Statement

This section presents the system design models (SW and HW models, constraints and the fault models) and the problem formulation.

4.1 System Model

The system model is decomposed into several models described as follows. The SW model presents the functional and *extra-functional* requirements of jobs and the HW model is the physical execution platform for those jobs. The fault model depicts the types of faults and their causes, whereas constraints define the possible solution space. The rest of this section details characteristics of the different models.

SW Model: The PIM of a hard RT application has to be enriched with the specification of temporal and dependability related requirements controlling its mapping to the PSM. This section presents the description of the properties of jobs and mathematical formulation in order to serve as a reference basis for the description of the PIM-to-PSM mapping algorithms, while Section 8.2 will address the SW technology context.

The designated functionality of the target system can be characterized by its respective HW resource demand, SW models, and the anticipated fault models associated with them. The SW model consists of a set of job types $\mathscr{J} = \{j_1, \dots, j_n\}$. Jobs represent the smallest executable SW fragments with basic communication capabilities for inter-job information exchange. Each job type j_i has the following attributes associated to it:

- *resource requirements* are summarized in a record (represented by the vector $\mathbf{r}_i$) composed of the different quantitative descriptors of resource capacity required, like CPU capacity, memory size, availability of a certain kind of sensors etc.
- *degree of criticality* (dc_i) measured as the number of the replicas needed for the particular job type.

Inter-job communication is characterized by a weighted directed graph (WDG), $G = (\mathscr{J}, E)$, having the job types as vertices V, and an edge between jobs j_s and j_t, if they communicate. Timing properties is represented as (t_i), which is the triple of $t_i(EST_i, CT_i, D_i)$, where EST, CT, D are the earliest start time, computation time and deadline of a job respectively. $e_{ij} \in E$ is an edge between two job vertices $(v_i, v_j) \in V$, which is the notion of both of influence (I_{ij}) and communication data $(b_{i,j})$ (bytes) between jobs. I_{ij} denotes the cumulated conditional probability of error propagation from the source job j_s to the target job j_t, either via message passing or shared resources, assumed, that j_s is in a erroneous state. $b_{i,j}$ is the amount of data of the required communication between jobs, for instance measured by the maximal total size of information to be transferred per execution cycle.

HW Model: We assume a distributed shared platform with a network topology allowing a HW node to communicate with each other node as shown in Figure 2. A HW node may contain a single or multiple processors or a processor with multiple cores. The set of nodes $\mathscr{N} = \{n_1, \dots, n_k\}$ can be modeled as an interconnection HW graph that represent limited HW capability provided by the node processor. The measure of limitation can be in time (e.g., a certain amount of CPU time is assigned) or in space (e.g., a certain memory region is assigned to a partition). The OS kernel layer supports the intra-node processor communication (e.g., by shared memory, buffer). For inter-node communication, nodes share the same communication channel to send and receive messages (e.g., by message passing). Jobs are mapped onto nodes which is represented as $\forall_{i,k} M(j_i, n_k)$, where i^{th} job j_i is mapped onto k^{th} node n_k.

Constraints Model: Constraints define the conditions that limit the possible mappings from a dependability, RT or resource perspective. A set of constraints $\mathscr{C} = \{c_1, \ldots, c_l\}$ need to be satisfied for a mapping to be valid [15]. Based on the requirements presented in Section 3.1, we summarize the following constraints: *(a)* binding constraints - jobs that need to be allocated onto specific nodes due to the need of certain resources (e.g., sensors or actuators), *(b)* FT constraints - separation of replicas to different nodes, *(c)* schedulability - maintaining RT constraints and *(d)* computing constraints - such as the amount of memory available for jobs.

Fault Model: We consider both SW and HW faults, therefore a fault can occur in any job, HW node or communication link. The consequence of a fault is an error (deviances from the functional or temporal specification) which can propagate from a source module to a target module explicitly via an erroneous message sent by a faulty job or via some shared resource (implicit propagation channel). A single (transient or permanent [31]) fault impacting any of these shared resources is likely to affect several or all of the jobs running on the node. In the case of communication link, only transient faults are considered.

4.2 Problem Formulation

The generalized resource allocation problem can be modeled as a *Constraints Satisfaction Problem* (*CSP*), which is characterized by a given set of jobs $\mathscr{J} = \{j_1, \ldots, j_n\}$, a distributed computing platform associated with k nodes $\mathscr{N} = \{n_1, \ldots, n_k\}$ and by a set of constraints $\mathscr{C} = \{c_1, \ldots, c_l\}$. A solution to this problem is an assignment of each of the n jobs to one of the k nodes such that all constraints $\mathscr{C} = \{c_1, \ldots, c_l\}$ are satisfied and objectives are met. The set of all possible mappings for a given set of jobs and nodes is called the *design space (X)* that includes feasible (X') and infeasible region ($X - X'$). The constraint surface (Figure 3) divides the design space into two regions: feasible and infeasible. Constraints that represent limitations on the behavior or performance of the system are termed as behavior constraints (e.g., FT and RT constraints) and that represent physical limitations are called geometric/side constraints (e.g., binding constraints) [34]. All these constraints are satisfied during the mapping algorithm presented in Section 6.4.

A hypothetical design space is shown in Figure 3, where the infeasible region is indicated by the hatched line. A *point x* in the design space X represents a mapping of jobs onto nodes. Points located in the region of constraints satisfaction are feasible points. A mapping is either feasible/aceptable or infeasible/unacceptable. A feasible mapping is a solution which satisfies all constraints $\mathscr{C}$. If any constraint is not satisfied then the mapping is infeasible. The *neighbourhood* space $N(x) \subseteq X$ of a point x is the set of all points that are reachable by performing a *move* operation (e.g., relocating a job to a different node). This parameter is used either creating an initial feasible mapping when backtrack is necessary or an optimized mapping both from feasible and infeasible one. Our mapping algorithm presented in Section 6.4 searches the global space X for a solution in the region of X'. It is a constructive heuristic which creates a feasible mapping for a set of jobs and nodes in every single run of the algorithm if a solution exists at all. Usually there exist many mappings that satisfy the defined constraints. Therefore measures are needed to find a suitable mapping. The *value* of a point is a measure of the suitability of the mapping represented by that point. The function $f(x)$ is used to measure the value of a point of the design space. For an optimization problem, which minimizes the value of objectives, good mappings have low values. The task is to find a mapping $x^* \in X$ with the lowest function value, i.e., $f(x^*) \leq f(x)\ \forall x \in X$.

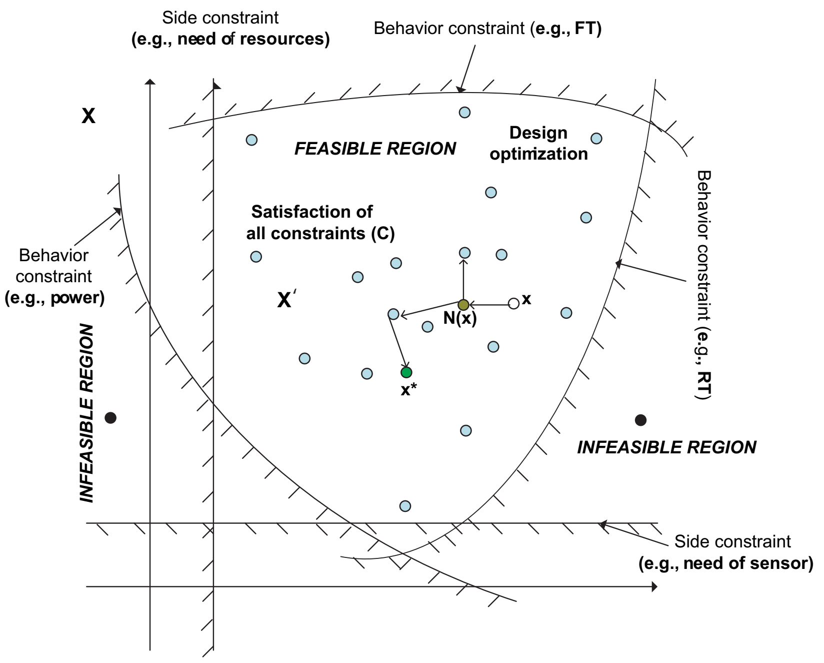

Side constraint
(e.g., need of resources)
Behavior constraint (e.g., FT)
X
FEASIBLE REGION
Design optimization
Satisfaction of all constraints (C)
Behavior constraint (e.g., power)
X′
N(x)
x
x*
Behavior constraint (e.g., RT)
INFEASIBLE REGION
INFEASIBLE REGION
Side constraint
(e.g., need of sensor)

Fig. 3 Hypothetical design space

x^* is the optimized mapping from the search space of X. However, guidance of heuristics is necessary for an efficient search in the global design space and for obtaining an optimized mapping with less computation cost. In order to prove this we have performed a comparative study with [35], where scheduling (ordering jobs execution) is implemented and conducted in a CPLEX based tool. CPLEX is an ILOG software product for solving Linear and Mixed Integer programming problems [36].

It is already mentioned that the problem is NP hard, therefore, in order to reduce the complexity we divide the process into subproblems. The mapping problem itself is divided into two subproblems: allocation and scheduling. First, we create a feasible allocation by using the proposed algorithm satisfying all the defined constraints including schedulability. The algorithm considers the proposed jobs and nodes ordering heuristics presented in Section 6.3 in its construction. The jobs and nodes are ordered before the allocation takes place which helps to find a feasible solution with less number of iterations (see experimental results in Section 7.2). During the allocation phase all the constraints are satisfied in a prioritized manner in order to be able to create a feasible mapping. If any of the constraints is not satisfied in *Step* 7 of the Algorithm 2 we perform the backtracking in *Step* 8. Moreover the assignment process is divided into phases according to the criticality of applications. In the first phase we consider only jobs from SC applications and after assigning them we consider jobs from non-SC applications. A validation test for the allocation is then performed so that it can be scheduled. The output of the algorithm derives the basic scheduling. An optimized solution can then be easily found by using CPLEX or any other approaches like [37]. The initial feasible mapping guides the optimization process in an efficient way to find the solution (see the validation and comparative study in Section 7.3).

5 The Transformational Approach

On the basis of the design aspects and system model presented in Section 3, we now briefly describe the system design flow within this transformational approach. Section 5.2 & 5.3 describe the consistency check of the input models used in the process and the constraint handling techniques.

5.1 System Level Design Flow

In this section, we describe the transformation based system design framework shown in Figure 4. The design process starts specifying the varied system requirements. These requirements can be captured for example by using the technique like in [38]. Once the requirements are specified they are modeled in PIMs. This high-level specification modeling is completely independent of underlying platform details. The process continues over setting the HW platform resources and performing the mapping through to the implementation. We assume that specification of PIMs and description of the candidate set of HW resources and services are available prior a mapping can take place. Essentially the PIM is modeled with the jobs properties of functionality, computation time, degree of criticality etc. The requirements from the SC applications are modeled in SC PIMs and non-SC applications are in the non-SC PIMs.

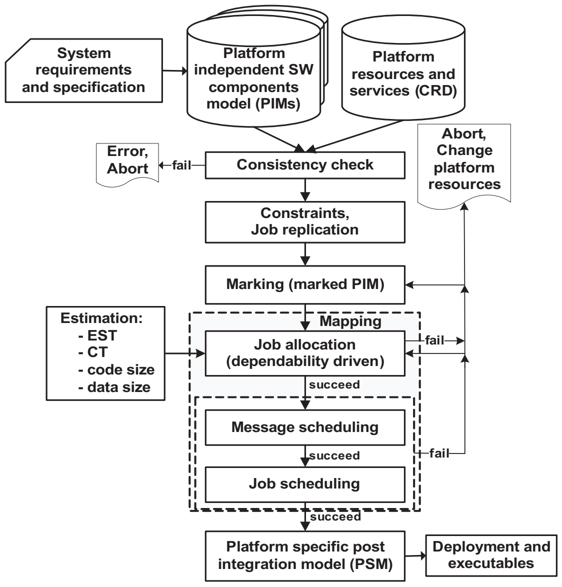

System requirements and specification
Platform independent SW components model (PIMs)
Platform resources and services (CRD)
Error, Abort
fail
Consistency check
Abort, Change platform resources
Constraints, Job replication
Marking (marked PIM)
Estimation:
- EST
- CT
- code size
- data size
Mapping
Job allocation (dependability driven)
fail
succeed
Message scheduling
succeed
Job scheduling
fail
succeed
Platform specific post integration model (PSM)
Deployment and executables

Fig. 4 Design flow for an integrated system design

As previously mentioned, properties that have to be satisfied in the mapping are modeled as *constraints*. All constraints imposed on application or platform level are extracted from the specification or defined by a designer before resource allocation can take place. This includes details such as timing information, memory and computational requirements. For SC jobs, designer has to specify the required degree of *replication* in order to ensure fault-tolerance. Other types of constraints, such as the computational capability and memory capacity of the computing nodes as well as network bandwidth have to be extracted from the platform details. The HW platform resources are modeled using CRD (Cluster Resource Description) [39] independent from the applications specification. It is represented using a meta-model called Hardware Specification Model (HSM) for capturing the resources of the platforms, e.g., computational resources, communication resources, special purpose HW like sensors or actuators etc. Based on this HSM meta-model, [39] develops a tool set for modeling the HW properties where designer can configure the resources according to their need. We have transformed this meta-model in the VIATRA framework in order to represent the quantity of each HW node (e.g., amount of CPU speed, memory, sensor/actuator etc.).

Before the mapping process can start, a consistency check of the input models (i.e., PIMs, CRD) is performed (see Section 5.2 for details), that means that checking the feasibility of transforming the input models into a platform specific post integration model (see Section 8 for details). As indicated in Figure 4 marked PIM (see Section 8.3 for details) is used in the design process in order to enhance the mapping process by complementing the information of PIM and HW platform model by designer decisions.

A crucial issue that comes up at this design stage is the mapping of jobs onto suitable nodes satisfying all the defined constraints.

A part of such job property descriptors dealing with type matching can be derived directly from their specification. For instance, a job delivering temperature values obviously needs a platform equipped with a thermometer of range, accuracy and sensitivity conforming to the specification.

Another part of job descriptors is related to the quantitative characteristics needed to the job-node allocation. For instance, information is needed in the form of parametrized job models on measures or estimates of job code, data size, timing requirements etc.

Here are two typical options depending on the level of readiness of the job implementation to be integrated:

In the case of the integration of an already complete job (as it is typical for reusing existing components) these parameters are available from prior measurements. Some characteristics like worst case execution times may need some simple adaptation to the particular candidate platform for instance due to variations in the processor speeds in different computing platforms built around the same processor types but having different clock rates. These types of information are provided in a parametrized form in the marked PIM.

Another case is when the overall system architecture design is performed concurrently with the integration design. Here expert estimates can substitute the temporally missing measurement results with a potential post-implementation iteration if the a posteriori measurements indicate an intolerable error in the initial estimator.

Once this information about models and jobs is obtained, the assignment of jobs onto suitable nodes is performed in the allocation phase applying the algorithm presented in Section 6.4.

The outcome of the allocation is used for scheduling which is performed in two phases as message and job scheduling:

– Message scheduling assigns a certain amount of bandwidth to each node and specifies the points in time of message transmission.
– Job scheduling is then performed satisfying their timeliness properties.

The infeasibility of the allocation or scheduling indicates an insufficiency in resources. If the time matching constraints cannot read satisfied than new times of platforms have to be introduced otherwise the capacity or the numbers of the nodes have to be increased.

The final task of the system level design is to deploy (integrating the PSM data with the application source code) and to create the executables for the target platform. The mapping process is elaborated in Section 6 and implemented in the VIATRA based tool set in Section 8.

5.2 Consistency Check

Consistency check is an important input filter of the process. It ensures that the input models are valid instances of their respective modeling languages. Usually, modeling language constraints are defined using the Object Constraint Language (OCL) [40]. During the design of the domain-specific languages for PIM and PSM, we also used OCL to express well-formedness criteria. During the language development phase, we discovered that ontologies can be used to check consistency of metamodels [41]. Using this technique, the domain-specific languages could be validated. As a byproduct, it has been shown that instance model completeness and consistency can also be validated by means of ontologies. Compared to OCL, the advantage of this technique is that it works both on meta and instance levels. Techniques like in [42] can also be used in order to formally verify the created models, e.g., verify the platform independent semantics by showing that the system under test conforms to the specification.

5.3 Constraint Handling and Design Optimization

The precise definition of requirements reduces the design space as they define constraints and objectives in addition to the designated functions, that limit the possible mappings from the dependability, temporal or resource perspectives. It helps to avoid the exploration of infeasible design alternatives [43], moreover a properly selected objective function may control an automated synthesis process to deliver (sub)optimal solutions. Applying the efficient search methods and techniques for constraints satisfaction we can avoid the unnecessary exploration of infeasible regions in the design space and effectively guide the search space. In the following we discuss how the constraints are handled during the mapping and describe different search techniques employed in the allocation phase of the design process. The constraints handling techniques are employed during the mapping algorithm to satisfy the constraints in a systematic manner. We also briefly discuss the optimization aspect.

Constraints Prioritization

The efficient management of a large set of constraints can rely on *constraints prioritization* [25]. Here the system designer assigns priority levels to the individual constraints estimating their order to guide the search process of the design space. This technique can be used for partitioning complex constraint systems into sequentially solvable blocks by exposing causal interdependencies between the individual constraints. For instance, the replication of SC jobs precedes (in a SC application) all other decisions on job allocation as is expressed by the topmost priority assigned to the related constraints. While generating a feasible mapping the constraints are checked as a priority basis to satisfy them on each time node assignment, i.e., an evaluation is performed for the assignment in order to increase the search efficiency. This assignment evaluation step (described in Section 6.4.1) tries to find a feasible assignment for each job without any backtracking. The backtracking search technique described below is applied in the mapping algorithm only when there is no feasible assignment found for a particular job assignment on the available nodes, i.e., a dead-end is reached. The backtracking is seldom needed when search procedures integrate prioritization of constraints together with the ordering heuristics.

Backtracking

This mechanism enables us to undo some previous assignments in case there is an inconsistency[5], i.e., no feasible assignment is possible with the current search path. The backtrack process goes back to the earlier assignments and changes them to the alternative feasible ones. One simple and easy backtracking mechanism is the chronological backtracking which systematically changes the most recent past assignment and tries alternative ones. If it is not possible then it respectively goes back to the next most recent assignment. We have implemented this technique which includes moves like *relocate* (relocating a job to a different node) and *swap* (swapping the nodes between two jobs). If there is no assignments left to undo, i.e., search reaches its initial state, then the mapping is infeasible and the process terminates.

Design Optimization

Different objectives are combined into a single composite function by applying weights expressing their importance, or reached by using multi-objective optimization techniques [37]. In this design process, optimization is primarily realized by using the ordering heuristics. We assign jobs according to the heuristic of reducing influences so that the algorithm can find a local optimized solution. However, there can be more than one feasible mapping and even more than one optimized mapping, so we need mechanisms to explore all the feasible regions and consideration of different variables for global optimized solution. In order to find a global or near-optimal solution a Multi Variable Optimization (MVO) approach [37] is used. We have used influence, scheduling length and bandwidth utilization as objectives. Given the prime focus on designing SC systems the quantification of influence is described in detail in this paper. The tool presented in [35] is used to compare the approach described in this paper and takes throughput (end-to-end deadline), robustness (number of failures), number of nodes, cost in the objective function during the optimization.

6 The FT+RT Driven SW-HW Mapping

Increasingly embedded systems functionalities are being implemented as SW. However the availability of physical resources is not necessarily such that each SW component (that is equal to a job in our terminology) can be allocated to its own HW node. The situation is limited by physical (space, size), weight and economic constraints. Therefore mapping of those SW components needs to be performed onto limited and shared HW resources. We develop a framework which systematically guides the mapping of jobs (SW) onto a shared distributed computing platform comprising of HW nodes. The main drivers behind the mapping are to provide *(a)* FT assuring a certain level of dependability desired by the user, *(b)* to enhance dependability by reducing the probability of error propagation, and, *(c)* to satisfy the timeliness properties (RT) through schedulability analysis. Other requirements and constraints, e.g., satisfaction of need of certain resources, desire to reduce the communication load on network etc. are also taken into account to ensure a valid suitable mapping.

We develop an iterative mapping algorithm presented in Section 6.4. The algorithm employs various mapping strategies together with the job and node ordering heuristics. Heuristics are used to create feasible mapping for a reduced number of backtrackings, or no backtracking if an optimal ordering can be obtained [44]. The idea behind our heuristics 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. For example a node attached with sensor/actuator

[5] An assignment when it does not violate any constraints is said to be consistent.

will be preferred at the beginning of node ordering so that a job needs sensor/actuator can be assigned without exploring further nodes. In the algorithm, we start by assigning the first job from the ordered list onto the first node from the ordered nodes and continue until all jobs have been assigned. While doing the mapping the constraints are checked in a priority basis to satisfy them on each time node assignment, i.e., an evaluation and a *consistency enforcing* [44] is performed for the assignment. The proposed heuristics and the algorithm are implemented (in VIATRA) in the allocation phase of the tool-chain.

6.1 Basis of the Mapping

We now outline the strategies that drive the PIM-to-PSM mapping considering both FT and RT constraints. On the basis of the requirements and models presented in Section 3, we start sequentially by discussing the strategies for ensuring FT, followed by discussions on the desire to reduce sensitivity to errors by influence reduction. Next, the schedulability analysis is discussed. The strategies presented here are subsequently employed in the mapping algorithm presented in Section 6.4.

6.1.1 FT Schemes

Traditionally FT predominantly utilized HW based redundancy, e.g., Multi-computer Architecture for Fault Tolerance (MAFT) [45], Maintainable Real-Time Systems (MARS) [46], XBW [47] and JAS 39 Gripen [48]. The active replication based FT is used in order to tolerate both permanent and transient faults. Usually multiple HW components/nodes are formed as a single unit called as *fault tolerant unit* (FTU) in order to tolerate either one permanent and/or one transient fault. When a node detects a fault, it falls silent and other replica nodes provide the necessary services. In these approaches adding a new function requires adding a new HW node which is needed to be further replicated to provide FT. Hence this method of redundancy incurs high HW costs for adding new functionalities.

Thus, in distributed hard RT systems, FT is usually achieved through active SW or timing redundancy. In case of active replication, critical SW components/jobs in the system are replicated and the replicas perform their services in parallel [49]. The technique employs replica deterministic agreement protocols, e.g., assure that all replicas start with the same initial state and perform the same computation. For timing redundancy, once there is a fault during the primary execution of a job it repeats the execution. The FT scheme presented in this paper ensures dependability through replication of jobs from SC applications. FT is provided by allocating replicas of jobs onto distinct 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 the jobs from an application may not be equally critical, all jobs from a single application do not need to be replicated to an equal level. The degree of replication of jobs is specified by the system designer based on the necessary level of criticality, e.g., derived from the safety integrity level or from the specifications of the system or from the experimental vulnerability analysis [50] results. If the user sets a criticality degree (usually based on the knowledge and complexity of the application) uniformly on an application, all the jobs from that application have to be replicated equally. Replication of critical jobs makes the system more dependable. However overprotection leads to brute replication that may in turn come at the expense of increased hardware cost, power and schedulability. Thus a suitable degree of criticality needs to be set for each application jobs.

We have also investigated different techniques complementing replication for FT, such as re-execution, checkpointing [51] or roll-back recovery and the interplay of these techniques [52]. These recovery techniques are based on timing redundancy. The desired FT techniques depend on the considered fault model and also depend on particular application requirements. If an application needs to tolerate a permanent fault it has to be replicated in spatial domain. On the other hand if it needs to tolerate only transient faults then re-execution or checkpointing would be sufficient given that deadlines are not violated. Figure 5 shows the trade-off between different redundancy based FT techniques such as spatial and temporal redundancy, where the system tolerates 2 transient faults. Prior to executing any FT schemes, the faults need to be detected. The fault detection process detects the existence of faults in the system either implemented with the FT schemes or implemented separately. Examples of fault detection techniques include signatures, HW watchdogs, assertions, comparators etc. The overheads in time for fault detection and recovery always need to be considered with the execution time of particular application job. As we see from the Figure 5 (a) that the

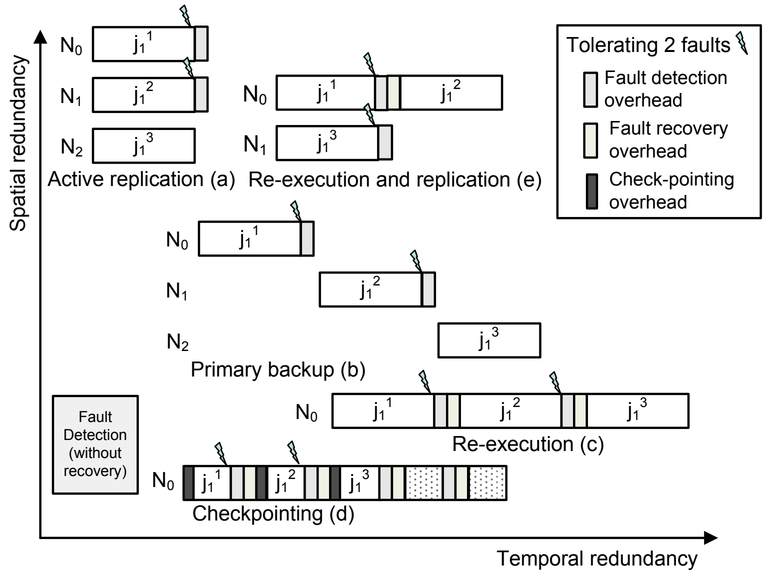


Fig. 5 Trade-off between different FT schemes (a)-(e)

SW based active replication uses more resources while taking less time to finish the computation. However this configuration tolerates permanent faults. Whereas techniques like re-execution and checkpointing (and combination with replication) use comparatively less physical resources, incur large time overhead and are only applicable for transient faults (Figure 5 (c), (d) and (e)). In the case of primary backup FT scheme (Figure 5 (b)) the main or the primary job run on a computing node and provide the services until there is a fault. If there is a fault in the system the backup replica starts executing on a different node and provide the necessary services.

In our approach we consider SW based active replication for tolerating both transient and permanent faults. The reasons for choosing active job replication over roll-back recovery or checkpointing is that while on one hand we consider tolerating permanent faults and on the other hand in hard RT systems roll-back recovery is often of limited use [43], due to, e.g.,: *(i)* as the roll-back/recovery can take an unpredictable amount of time, it is difficult to guarantee the deadline after the occurrence of a fault, *(ii)* an irrevocable action which has been effected on the environment cannot be undone, *(iii)* the temporal accuracy of the checkpoint data is invalidated by the time passed between the checkpoint time and the instant now and *(iv)* usually a perfect fault detection mechanism and permanent fault free data storage are assumed which may not be the case.

6.1.2 Influence Reduction

We strive to minimize the interactions and influences between jobs and also the communication load on the physical network by allocating jobs with the highest mutual communication onto the same node. *Influence* is defined as the probability of error propagation from a source to a target module. Faults can occur either in the source module or in the communication channel. The consequence of fault is an error. Shared memory is a typical element potentially causing error propagation between different functionalities. This propagation depends on the size of the memory they share and how often they access it. Errors can also propagate through message passing which depends on the size of the sending/receving messages and how frequently messages are being sent and received. All these error propagations between any modules are termed as influence. At an early design stage we are not necessarily aware of specific execution environments or communication protocols. Hence, the worst case scenario is assumed in order to design a system for a better cost-performance ratio by giving the provision of using a less efficient error detection mechanism or less efficient communication protocol.

The allocation of highly communicating jobs to the same node is beneficial both for reducing the bandwidth and for confining the inter job error propagation within a single node. In doing this, it is important not to violate FT, RT and resource constraints. The communication clustering heuristic, which attempts to allocate highly communicating jobs to the same node, thus reducing the overall communication load on physical network, has been addressed in [53]. As we consider design of an integrated system where several nodes share a single network, the communication clustering heuristic is desirable. Between two communicating/interacting jobs, there is an influence that may lead to propagation of errors from one job to the other. When communication between two jobs is high, the influence 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 influences risk the failure of multiple nodes and are undesirable. Moreover, messages sending over the network can cause loss of messages due to transmission error, e.g., in automotive cars electro-magnetic interferences causes communication failure due to transient errors.

Example Describing the Benefits: We consider an example of an application (similar to [54]), which consists of four jobs j_1, j_2, j_3, and j_4 and need to be mapped onto an architecture consists of two nodes (n_0 and n_1) communicating via a network. The application and the architecture is shown in the upper part of the Figure 6. All jobs must finish their execution by 140ms, i.e., by the deadline of the application. Individual CTs for each job are shown in the figure, e.g., job j_1 takes 40ms for its execution. A particular job takes the same amount of CTs to execute on either processor. j_1 is a predecessor of j_2 and j_3, and sends messages m_{12} and m_{13} to j_2 and j_3 respectively. j_4 is a successor of j_2 and j_3, and receives messages m_{24} and m_{34} from j_2 and j_3 respectively. A TDMA based network is assumed for the communication where a TDMA round TD_x comprises of two slots s_0 and s_1. For the purpose of deterministic message transmissions node n_0 and n_1 are statically assigned to slot s_0 and s_1 respectively. The slot length of the network is equal to 10ms and maximum 2 messages can be sent per slot. The time for intra-communication (communication within the same node) is assumed to be zero. This is shown in Figure 6 (b) when j_1 and j_3 are allocated on node n_0. These two jobs communicate through the services provided by the OS

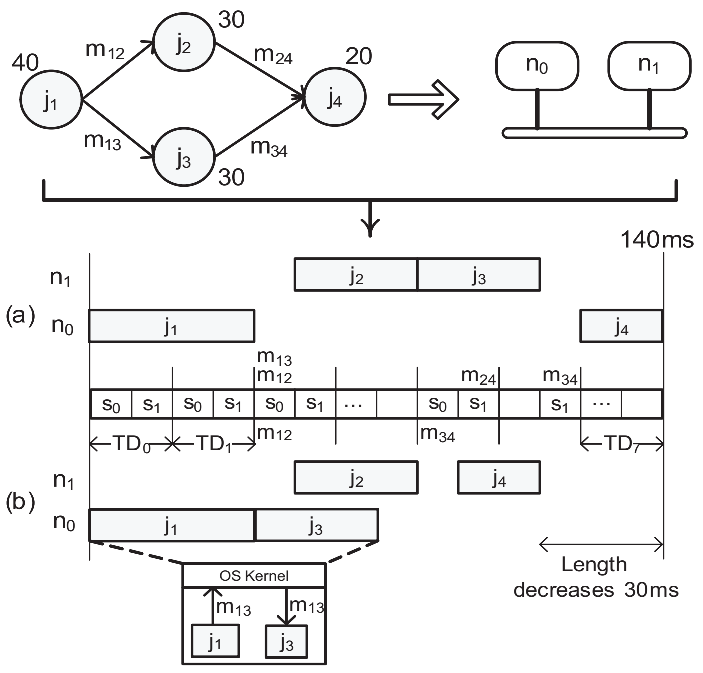


Fig. 6 Reduction of influence and communication overhead

kernel layer while taking a negligible amount of time comparing with the time taken by the communication channel. We assume that the partition switching time is also negligible.

Figure 6 illustrates that in a typical case, allocating the interacting jobs to the same node (case b) is resulting in a reduced end-to-end delay and network load, in compared to a generic allocation pattern (case b). We emphasize the following key benefits (where first two benefits enhance dependability) of assigning highly interacting jobs to the same node:

(1) Restricting the possible nodes from correlated faults,
(2) The probability of losing messages over the network is reduced,
(3) The communication load on the network is reduced (may allow for the use of a slower but cheaper bus [17]) and
(4) Increases the over all performance by reducing the total execution time (computation time + time to send/receive messages) of a job since network delays are avoided.

Estimating Influence

Deviances from the correct state corresponding to the specification, or in other words *errors* originate either in some local fault of a component or in corrupted measseges. They may propagate along the messages.

Influence covers three phases of error propagation as shown in Figure 7 (a), namely: *(1)* a fault/error occurring in a module or in a communication link, *(2)* propagation of the fault/error to another module and *(3)* the propagating fault/error causing a cascaded error in the target module.

In order to quantify influences, we assume P_e as the probability of error propagation from source to target considering no corruption over the network and P_l as the probability of message corruption over the network. If the message size is large, or the frequency of sending messages is high, then the probability of messages getting corrupted over the network is high.

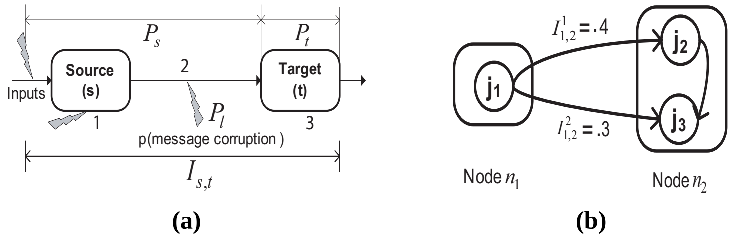


Fig. 7 (a) Phases of error propagation and (b) Combining influences

The probability of error propagation from a source (s) to a target (t) is denoted by $P_{s,t}$ and defined as follows:

$$\begin{aligned} P_{s,t} &= p\{error\ propagation|no\ corruption\ over\ the\ network\}* \\ &\quad p\{no\ corruption\ over\ the\ network\} \\ &= P_e \cdot (1-P_l) \ = \ P_s \cdot P_t * (1-P_l) \end{aligned} \tag{1}$$

Where,

$$\begin{aligned} P_e &= P_s \cdot P_t \\ P_s &= p\{error\ in\ output\ of\ s|error\ in\ input\ of\ s\} \\ P_t &= p\{error\ in\ state\ of\ t|error\ in\ input\ of\ t\ coming\ from\ s\} \\ P_l &= p\{message\ corruption|error\ on\ the\ communication\ link\} \end{aligned}$$

The probability that s outputs an error and sends it to the input of t is P_s. The probability that an error occurs in t due to the error received from s is P_t. The former indicates how often s allows errors to propagate out of s and the latter indicates how vulnerable t is to errors propagating from s. The probability of message corruption P_l can be defined as the unreliable message transmission over the network, which is calculated as $1-exp(-\lambda_l \cdot \frac{b_{s,t}}{T})$. Where, $b_{s,t}$ is the size of the messages between s and t and T is the transmission speed. Assume that the failure rate of the communication link is λ_l. $exp(-\lambda_l \cdot \frac{b_{s,t}}{T})$ is the reliability factor due to message transmissions over the network, i.e, the probability that the messages are transmitted safely.

We further elaborate on different error probabilities. An error (e.g., a bit flip transient error) occurs at any inputs of s or generated from any other sources and may propagate to input of t, where an error may occur. The probability of an error in I_y (the y^{th} input or the y^{th} source to propagate out of s) is $P_s^{I_y}$ and is expressed by $0 \leq P_s^{I_y} = p\{s|I_y\} \leq 1$. If there is more than one input or error sources, the equation is generalized [50] for calculating the error transmission probability P_s:

$$P_s = \sum_{y=1}^{Y} (p\{I_y\}/Y) * p\{s|I_y\} \tag{2}$$

Where Y is the number of inputs of s and $p\{I_y\}$ is the probability of occurring error in inputs or in any other sources.

Measuring by Fault Injection: We now describe an experimental estimation of influence using *fault injection*. The error propagation probability is estimated using the following procedure: *(a)* in each input I_y of s inject an error (one input at a time, i.e., no multiple errors), *(b)* observe the state and output signals of s and the state and outputs of t, and *(c)* use golden run comparison (i.e., comparing an injection run with a *golden* reference or the fault-free run) in order to detect when errors have occurred in either. Let the number of injection runs where errors in the output of s and in the state and output of t have been detected be denoted as $\eta_{err,s}$ and $\eta_{err,t}$ respectively. The total number of injection runs is denoted as η_{inj}. We then estimate the error probability as $P_s = \frac{\eta_{err,s}}{\eta_{inj}}$ and $P_t = \frac{\eta_{err,t}}{\eta_{inj}}$.

Overall System Level Influence: Considering both $P_{s,t}$ (comprises P_s and P_t) and P_l, the influence for a single error propagation path is calculated as $I_{s,t} = P_{s,t} + P_l$. The overall influences between a set of jobs assigned together on a node and interacting jobs allocated on different nodes is denoted as $I^o_{s,t}$ and expressed as:

$$\begin{aligned} I^o_{s,t} &= 1 - (1-I^1_{s,t})\cdot(1-I^2_{s,t})\cdots(1-I^x_{s,t}) \\ I^o_{s,t} &= 1 - \prod_{\rho}(1-I^{\rho}_{s,t}) \end{aligned} \tag{3}$$

where $\rho = 1,\ldots,x$ is the number of influences paths between two modules.

We consider the following example shown in Figure 7 (b) where a job j_1 is assigned to a node n_1, and another two interacting jobs j_2 and j_3 are assigned onto n_2. The overall influence of node n_1 to n_2 will be: $I^o_{n_1,n_2} = 1-[(1-0.4)\cdot(1-0.3)] = 0.42$.

Influences are assumed to be zero for jobs which are assigned on the same node, e.g., the influence between j_2 and j_3. If all these three jobs could be assigned onto a single node then the error would contain within that node only. However it is not possible to assign all interacting jobs onto a single node due to imposed constraints. Also replicas need to be placed on different nodes which might have influences with other jobs. Hence, there will be jobs interacting across nodes. We strive to minimize these influences as much as possible for a mapping such that dependability is enhanced by design. Values for error occurrence probabilities can be obtained, for example, from field data or from system specification or by fault injection [50]. The computation of the system level influence $\hat{I}_f$ is expressed as follows, which is then normalized, where k is the number of nodes: $\hat{I}_f = \sum_{i,j=1}^{k} I^o_{i,j}$.

6.1.3 Schedulability (RT) Guarantee

Once the jobs have been allocated to different nodes, the scheduler takes over the task for generating the execution sequences of jobs. In our approach, the timing constraints specified in Section 4.1 are checked during the assignment of jobs, i.e., in the allocation phase in order to ensure that the mapping is schedulable. When assigning jobs to a processor which already hosts one or multiple jobs, precedence and deadlines constraints are checked to ensure schedulability. Such a check was developed in our previous work [15]. However the deadline of a job sending a message to another job located on a different node must be reduced by the time for transmitting a message across the bus to accommodate for possible network delays (T_N) (see Figure 8) and the precedence relations have to be preserved as well. In this paper, we particularly focus on communication/message scheduling and provide a measure of network delay by using the *time-division multiple access* (TDMA) protocol as an example.

Scheduling for Integrated Systems: A hard RT system must execute a set of jobs in such a way that time-critical jobs meet their specified deadlines [43]. In traditional system design one function is assigned to a single node and therefore, typically, application jobs are scheduled on an independent processor, i.e., uniprocessor scheduling. Such types of scheduling and schedulability analysis have been discussed in [55]. The principal assumption made is of scheduling independent tasks/jobs onto a single processor. However in our integrated system design, jobs from different criticality applications are assigned onto a single processor and jobs from single application are assigned onto different processors, i.e., a multiprocessor scheduling. Moreover jobs usually have precedence relations among each other, i.e., the execution of one job depends on the result from the other. Consequently, new schedulability analysis techniques are needed [1] which can consider distributed applications, data and control dependencies, and accurately take into account the communication protocols that have a significant influence on the timing properties. For such distributed RT systems, specifically the type of systems whose failure can be catastrophic due to violation of deadlines, static scheduling algorithms are used to build, off-line, schedulability tables with activation times for each job such that timing constraints are satisfied. TTP/TTX-Plan (for TTP/C and FlexRay networks respectively) and TTP/TTX-Build are commercial tools [56] employing such scheduling techniques. Using the time-triggered communication protocol (TDMA as communication scheme), TTP/TTX-Plan and TTP/TTX-Build derive the off-line schedule for messages and jobs respectively. They are utilized in our developed tool-chain to generate the scheduling. In order to validate the schedulability analysis we also use the alternative tool [35].

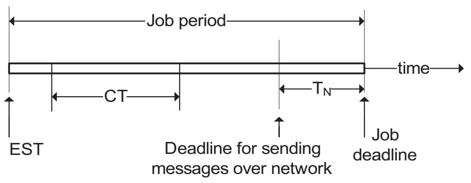


Fig. 8 Network delay (T_N)

Message Transmission Time: For communicating jobs located on different processors the message transmission time through the network has to be considered to make the jobs schedulable. A maximum network delay T_N is assumed 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 in this case 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. This is also utilized in the TTP/TTX-Plan message scheduling tool. A similar message communication planning can be found in [54] where the authors determine the slot and the round for a specific message to be sent. In Algorithm 1, the estimation of T_N as well as the actual EST of allocated jobs are provided. The following expression is necessary to calculate ESTs of jobs having precedence relations:

$$EST_i \geq EST_j + CT_j + T_{N_{i,j}} \text{ (job } i \text{ depends on job } j\text{)} \tag{4}$$

where $T_{N_{i,j}}$ is the network delay due to message transmission between jobs j_i and j_j.

We use following parameters in the Algorithm 1. n_k is the node which is ready to send the message over the network and s_k is a specific slot assigned to it. b_{m_i} is the size of the

Algorithm 1 Network delay calculation for messages transmission

```
1: Function: message_transmission($n_k, b_{m_i}, release\_time$)
2: $s_k$ = assigned slot to $n_k$; /*Slot assigned in TDMA round*/
3: $round = floor(release\_time/round\_length)$; /*Calculate current round*/
   /*Next step checks whether the slot of the current round has passed*/
4: if $release\_time - round * round\_length > start_{s_k}$ then
5:     $round = round + 1$; /*Increase a TDMA round*/
6: end if
7: while $b_{m_i} > b_{s_k} - b_{occupied}$ do
8:     $round = round + 1$; /*Increase round if the message size does not fit in this slot*/
9: end while
10: $T_{n_i} = round * round\_length + start_{s_k} + slot\_length - release\_time$; /*Calculate the delay for a job sending
    messages over the network*/
11: return($round, s_k, T_{N_i}$);
12: end message_transmission
```

message ready to send at time *release_time*. Therefore, *release_time* is the message delivery time of a job. b_{s_k} is the size of corresponding slot where as $start_{s_k}$ is the starting time of the slot in a round. For a successor of more than one jobs, maximum delay caused by all its precedence jobs is used for calculating the actual EST and is computed as follows:

$$EST_i = max\left\{ready(j_i, n_k),\ max_{\forall j_j \in pred(j_i)}\left(ft(j_j, n_p) + T_{N_{i,j}}\right)\right\} \tag{5}$$

where $ready(j_i, n_k)$ is the earliest time at which processor n_k is ready to start executing the job j_i. $pred(j_i)$ is the set of all predecessor jobs of j_i. $ft(j_j, n_p)$ is the finish time of job j_j in node n_p computed by the sum of EST_j and CT_j. The message transmission delay between (j_i, n_k) and (j_j, n_p) is $T_{N_{i,j}}$. The scheduling length of a mapping can be calculated using the following expression:

$$S_l = max_{\forall(i,k)}\left\{ft(j_i, n_k)\right\} \tag{6}$$

6.2 Supporting Data Structures

We introduce matrices for the purpose of ease structuring and implementation of the mapping algorithm presented in Section 6.4. The *allocation compatibility 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 is used to determine the most communicating jobs.

Allocation Compatibility Matrix A**:** A rectangular matrix $A_{k\times n}$ is used to describe possible assignment of a single job onto nodes, in such a way that rows represent nodes and columns represent jobs, where k 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 cannot. Restrictions on which nodes a job can be assigned to is the result of binding constraints and are determined by the use of particular resources, e.g., when a job needs sensors or actuators.

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 mutual 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 for one time execution (i.e., the available size of sent and received messages as defined by the system user).

6.3 Ordering Heuristics

Assignment of jobs and nodes needs two important heuristics for effective solving of the mapping problem, namely how to decide which job to assign next (ordering of jobs), and which node to assign to this job. This is similar to the so called variable (job) and value (node) ordering heuristics which are concerned with the order in which variables are instantiated and values are assigned to each variable. A good variable ordering is one that starts with the variables that are the most difficult to instantiate (i.e., 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., a so-called least constraining value ordering heuristic). In the next section, we provide details on this using an example. These heuristics can have a significant impact on search efficiency. No backtracking would be necessary if an optimal variable/value ordering is achieved [44]. Thus in such a case a linear time solution for the mapping problem is possible. Therefore a proper and good selection of ordering can reduce the number of steps to find a solution. For creating the mapping, we propose job and node ordering heuristics which are described in the subsequent sections. Of course the proposed ordering is no guarantee that backtracking will never be necessary. However we believe that the proposed *a priori* heuristics for ordering jobs and nodes is a viable strategy as a justification of ordering heuristics is provided in the next section.

6.3.1 Justification of Ordering Heuristics

The algorithm presented in the next section considers the most conflicting jobs that cannot be mapped on the same node, e.g., replicas and the most important jobs, e.g., binding functionality of jobs first which can easily guarantee the feasibility of the generated initial solution. This consideration is realized by explaining the following example.

Let us consider the following jobs j_1, j_2, j_3 and j_4 and their assignment onto nodes n_1 and n_2. Job j_2 is a high critical job and is replicated twice j_{2a}, j_{2b}. We assume that two jobs can run on one node and three jobs can run on another node with sufficient resources. Four constraints are defined which need to be satisfied during the assignment: c_1- due to the binding functionality j_1 must run on n_1, c_2- j_{2a} and j_{2b} must run on separate nodes in order to tolerate faults, c_3- computational capability and c_4- memory resource capacity. The type c_3 and c_4 are common/general constraints that are always present in each allocation problem. They refer to all jobs and therefore do not give us a direct hint which jobs should be assigned first. c_2 explicitly refers to conflicting jobs, i.e., replicas j_{2a} and j_{2b} and c_1 exclusively mentions about j_1 which requires a sensor. Thus, j_1 and j_2 are to be considered first for the mapping. Therefore, we start by assigning j_1, j_2 and then j_3, j_4 as follows: *(a)* assign j_1 onto n_1 as enforced by c_1, *(b)* j_{2a} and j_{2b} are assigned to n_1 and n_2 due to constraint c_2, *(c)* j_3 can now be assigned to either node, we arbitrarily choose n_1 and *(d)* j_4 must be assigned to n_2 (due to the resource constraints of the nodes). Now, assume an arbitrary job ordering of $j_1, j_3, j_4, j_{2a}, j_{2b}$. The assignments are as follows: *(a)* assign j_1 onto n_1 as enforced by c_1, *(b)* j_3 can be assigned to either node, we arbitrarily choose first node n_1, *(c)* j_4 can be assigned to any node, we choose n_1 again, *(d)* as enforced by c_2, j_{2a} and j_{2b} have to assign on different nodes but we fail to assign on n_1 due to c_3 and c_4, therefore, *(e)* repeated backtracking (or back jumping to step *(b)* and *(c)*) is necessary so that either j_3 or j_4 can be moved to n_2 to create a feasible assignment. When job j_1 is considered last in the order list backtracking is also necessary.

6.3.2 Job Ordering-Heuristics

These heuristics are used to order the jobs, i.e., to decide which jobs to assign first. The compatibility matrix A and the communication matrix C is also employed 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 a *Phase*[6] 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. By considering these jobs first, the search space is likely to decrease since these jobs are the most constrained (with respect to binding constraints). Ties are broken according to the second heuristic given below.

2a. 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 II*, a sub-matrix $\widetilde{C}$ of the communication matrix C is created, 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 I*. The reason for including already assigned jobs in the matrix $\widetilde{C}$ in *Phase II*, is that jobs to be assigned in this phase belong to SC PIMs and thus are more likely have communication with the jobs previously assigned in *Phase I*.
 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 compatibility matrix A nor the full communication matrix C. Just the sub-matrices suffice. Further the sub-matrix of the communication matrix, which is used in *Phase I* is itself a part of the sub-matrix used in *Phase II*. Hence, the sub-matrix of *Phase II* could be created and used in *Phase I*, reducing the number of matrices that need to be created. Also, the symmetry of the communication matrix (and its sub-matrices) can be exploited in the implementation. In this case the search for the highest mutual communication pair relies either only upper or lower triangular part of the matrix $\widetilde{C}$ as shown in Table 2.

6.3.3 Node Ordering-Heuristic

Just as in the job ordering-heuristics, the same sub-matrix $\widetilde{A}$ of the compatibility matrix A is used 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.

6.3.4 Example Describing the Ordering Heuristics

We take an example which consists of four jobs j_1, j_2, j_3, j_4 and two nodes n_0, n_1. Consider that job j_2 needs a sensor and node n_1 has a sensor attached to it. Let us assume that the mutual communication volume within a time period between j_1 and j_2 is 4 bytes, between j_1 and j_3 5 bytes, between j_2 and j_4 8 bytes and between j_3 and j_4 is 5 bytes. When

[6] Phases (I, II and III) are used in the Algorithm 2 described in Section 6.4.

the assignment compatibility matrix A is created (Table 1), we see that job j_2 can only be assigned to node n_1, correspondingly n_1 is the only usable node for j_2. Since j_2 cannot be assigned onto n_0, we put 0 in the corresponding cell. All other jobs can be assigned to any of the two nodes. We put 1 in the corresponding cell when a job can be assigned onto a node. The ordering of jobs will start by j_2 and node n_1 will come first in the node

A	j_1	j_2	j_3	j_4	$\sum$
n_0	1	0	1	1	**3**
n_1	1	1	1	1	**4**
$\sum$	**2**	**1**	**2**	**2**	

Table 1 Building assignment compatibility matrix

C	j_1	j_2	j_3	j_4
j_1	**0**	4	5	0
j_2	4	**0**	0	8
j_3	5	0	**0**	5
j_4	0	8	5	**0**

Table 2 Building communication matrix

ordering. Ties are broken for other jobs by using the communication matrix (Table 2). Each cell of the communication matrix is filled by the total amount of mutual communication volume in bytes of the corresponding jobs pair. The upper and lower triangular parts of this matrix is symmetric. Hence during the implementation we only need to search for the communication pairs either in upper or lower triangular part. We see that job j_2 and j_4 have high mutual communication among all the pairs followed by the pair j_4, j_3 and j_3, j_1. Hence the job ordering will be j_2, j_4, j_3, j_1 and the node ordering will be n_1, n_0. If a job appears more than once in different pairs then from the first pair it is placed in the ordered list. If we assume that maximum two jobs can be assigned on a single node due to the computation and resource constraints then node n_1 will host the jobs j_2 and j_4 and n_0 will host the jobs j_3 and j_1. The replicas are not included in either matrices only the primary job is included. When a job is replicated two times the corresponding communication link of the job is also replicated and the communication volume becomes double. Only the primary replicas (the main job) are considered in the communication matrix for their ordering. The other replicas are not considered as they will be anyway assigned onto different nodes and will disseminate messages over the network.

6.4 The Algorithm

The construction of the algorithm is inspired by the established constructive heuristics in space allocation [57], in course timetabling [58] and by the variable and value ordering-heuristics for the job shop scheduling constraint satisfaction problem [44]. The algorithm works in three phases and considers SC PIMs and non-SC PIMs separately to reduce influences. As a result of component based design, SC and non-SC PIMs communicate minimally, thus they can be treated separately. To facilitate strong partitioning between SC and non-SC PIMs, we allow that jobs of SC PIMs and jobs of non-SC PIMs can be allocated onto separate processors or cores on the same node. The jobs are assigned in three different phases, mentioned below:

Phase I: High critical jobs of SC PIMs,

Phase II: Non-replicated jobs (if any) of SC PIMs and

Phase III: Jobs from non-SC PIMs.

The described 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

Algorithm 2 Extra-functionality driven SW-HW mapping algorithm

Input: J :set of jobs, N : set of nodes
Output: $alloc$: set of job to node mappings

```
1: $J$ :={all jobs to be assigned in this phase}
2: $J := replicateJobs(J)$ /*Replicate jobs according to their degree of criticality*/
3: $J := orderJobs(J)$ /*Order the jobs according to a job ordering-heuristics*/
   /*Section 6.3 provides a discussion of the heuristics used.*/
4: $N := orderNodes(\{allnodes\})$ /*Order all nodes according the heuristics*/
   /*Section 6.3 provides a discussion of the heuristic used.*/
5: $j_i := getFirstElement(J)$.
6: $n_k := getNextNode(N, j_i)$ /*$n_k$ is the next node that has not been evaluated already
   as a possible assignment for $j_i$.*/
7: if $n_k = null$ then
8:     if $alloc = \emptyset$ then
9:         $fail$/*allocation is not possible*/
10:    end if
11:    $(j_{last}, n_{last}) = getLastElement(alloc)$
12:    $alloc := alloc \setminus (j_{last}, n_{last})$/*delete last allocation*/
13:    $J := J \cup j_{last}$/*re-add last job to the set*/
14:    $goto$ step 5.
15: end if
16: if $assignmentValid(j_i, n_k)$ then
17:    $alloc := alloc \cup (j_i, n_k)$/*add new allocation*/
18: else
19:    $goto$ step 6.
20: end if
21: $J := J \setminus j_i$
22: if $ifJ \neq \emptyset$ then
23:    $goto$ step 5
24: end if
25: $stop$/*allocation completed.*/
```

respect to binding constraints) are assigned first. Using this ordering and assigning replicas in *Phase I*, the number of backtracks are reduced (see experimental results in Section 7.2). In *Phase II*, we continue with non-replicated jobs of SC PIMs, they will be integrated with the replicated jobs of SC PIMs in a way that reduces job influences. As the lower critical jobs from SC PIMs are treated in a different phase, it is more likely that there will be less influences between them. Finally, jobs from non-SC PIMs are allocated in the third phase. A high level description of each mapping phase is outlined in Algorithm 2. A detail description of $Step$ 7 and 8 of the algorithm is given in the following section. Section 6.4.2 describes the adaptability of the algorithm onto a heterogeneous platform.

6.4.1 Assignment Evaluation and Consistency Enforcing

Before a job can be assigned to a node, an evaluation has to be performed. In this step of mapping, all the defined constraints are satisfied. If the node is empty, i.e., there are no previously assigned jobs to that node, then only binding constraints need to be checked. If there are already assigned jobs on a node then apply *retrospective techniques*. Retrospective techniques are characterized by the assignment of a job to a node while checking other jobs that are already assigned in this node in order to avoid conflicts. If all constraints hold, i.e., consistency enforcing is ensured, then the next job is selected. This technique allows us not to try to assign the replicated jobs on the same node and is also enforced by the FT constraints. While assigning jobs during *Phase I*, different nodes are selected for the replicas in $Step$ 6, which significantly reduces the number of iterations as well as backtracks to find the feasible solution. The assignment process does not know *a priori* before checking the constraints whether there is a replica already assigned. When a replica job is chosen to be assigned from the list, a new node is selected for it to be assigned. If the verification of consistency fails, exploration continues with the next node. When all nodes for this job has been checked unsuccessfully, the backtracking goes back to the most recently instantiated job, and so on ($Step$ 8 in Algorithm 2). Backtrack is performed by simply swapping or moving the jobs between nodes. After performing a move, if a feasible solution is found then the algorithm is continued with selecting the next job from the actual list. If a solution is not found after backtracking then the algorithm returns an infeasible mapping.

In the case when non-SC applications share the same processor as SC applications, some optional strategies are possible. After the jobs belonging to SC applications have been assigned, nodes can be re-ordered in a way that eases the assignment of non-SC jobs. As an example, ordering the jobs according to the amount of remaining computation capacity of each node thus better *load balancing* between nodes can be achieved. Another possibility is to re-order the nodes according to least memory utilization or nodes having less failure rate (useful when heterogeneous platform is assumed). All of these re-orderings might also be beneficial from a dependability viewpoint. Since non-SC jobs will be primarily assigned to nodes with few/no jobs from SC applications, the separation of SC and non-SC jobs is likely to increase. This will reduce the likelihood of errors occurring in non-SC applications propagating to SC applications.

6.4.2 Applicability on Heterogeneous Platform

We explain the adaptability of our algorithm onto heterogeneous platforms in terms of computation and failure rates. A heterogeneous platform may consist of node processors of different speeds (f) and of different failure rates (λ_k). Therefore each job may have different CTs on different nodes and also the failure probability can be different. A processor with less failure rate is obviously more reliable and more jobs are assigned onto this node so that the system reliability is maximized. However, this may overload some processors while other processors are less utilized. In order to tackle this, heuristics like in [59] can be utilized while applying the retrospective technique in $Step$ 7. The technique is to *allocate jobs onto a processor according to the product of the failure rate and the total time (T_t) required by the instantiated job and already assigned jobs on that processor*. Assign the selected job on a processor where the product $\lambda_k \cdot T_t$ is minimum. We term this as $HetHeus$. This heuristic ($HetHeus$) provides a better trade-off between reliability and schedulability of the mapping. Moreover, if a job needs certain level of reliability and only a specific processor can provide it then this job should be assigned on that processor. These types of requirements can be taken into account in the allocation compatibility matrix A described earlier.

7 Evaluation of the Mapping

In order to demonstrate the effectiveness of the approach, we provide illustrations of different scenarios and the proof of concept of the algorithm using mixed-critical applications of actual automotive system from industry partners. Several experiments have been carried out to evaluate the performance of the mapping heuristics. The results are discussed in Section 7.2. Assuming a homogeneous platform the detail assignment policy by using the proposed ordering-heuristics has been described in [15]. In the following section, we are interested to show the assignment policy described in the algorithm onto a heterogeneous platform. This platform consists of node processors of different computing power and failure rates. A discussion of how the objectives are quantified to select a good mapping is depicted as well. At the end of this section we present the validation and a comparative study of the algorithm using an independent tool [35].

7.1 Mapping Illustration

We consider a mixed-critical application consisting of brake-force control (BFC) of adaptive cruise control (SC application) and mirror movement functionality (MVF) of doors control subsystem (non-SC application). For the sake of simplicity a part from each of these subsystems is considered as shown in Figure 9. The interaction from BFC to MVF comes from the fact that when the speed of the car is high the door must be locked. As the MVF is a part of doors control a communication is set so that when the speed is high mirror movement should not happen. However there is no data and control flow from the non-SC (MVF) to SC application (BFC) as error occur in non-SC application may propagate to SC application which is strongly prohibited in our design.

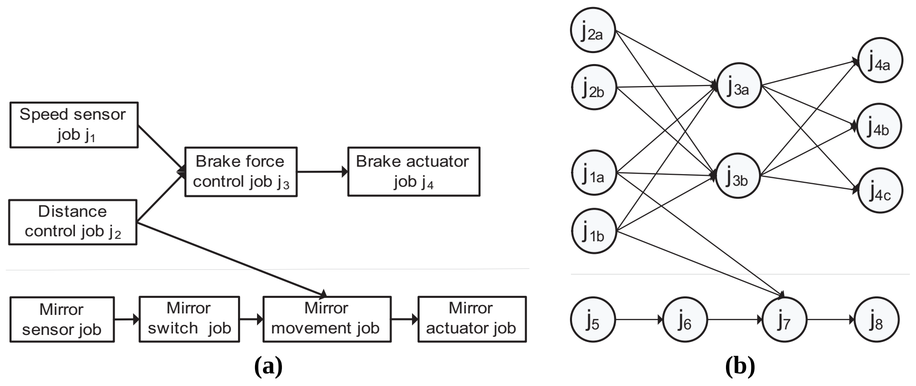


Fig. 9 (a) Mixed-critical application (SC BFC and non-SC MVF) and (b) after FT scheme

Properties of Jobs and Nodes: BFC is a video-based emergency brake for the collision warning and avoidance system. This subsystem is decomposed into four jobs j_1, j_2, j_3 and j_4 as shown in the upper part of Figure 9. Jobs j_{1a} and j_{1b} are the replicas of j_1 and the similar type of notation is used for all other replicas. Job j_1 is responsible for reading car speed value from the speed sensor and sends the speed message m_{13} to j_3. j_3 is a control object job for computing the necessary brake force. Job j_2 reads distance value of the nearest object from the image sensor and sends the corresponding value via a message m_{23} to j_3. Job j_3 computes the brake force and transmits the message m_{34} to brake actuator (BA) job j_4. j_4 activates the brakes in order to make the necessary actions to avoid collision. All the functional and extra-functional properties are modeled in a SC-PIM. The assumed job properties are as follows: Replication factor: $2,2,2$ and 3; EST : $0,0,20$ and $45ms$; CT : $20,20,25$ and $15ms$ for jobs j_1, j_2, j_3 and j_4 respectively. The values of CTs are adapted according to the speed of the node processor where the jobs are assigned. The influence value (calculated using the method defined in [50]) between j_1, j_3 and between j_2, j_3 is assumed as 0.40 and between j_3, j_4 is 0.30. The message size between each pair of job is 25bytes. All jobs from the subsystem have to finish execution by their deadline equal to the period of $150ms$. The chosen FT schema tolerates one failure (either transient or permanent).

The MVF is decomposed into 4 jobs j_5, j_6, j_7 and j_8 as shown in the lower part of the Figure 9. Job j_5 is a mirror sensor job, which reads the left mirror movement command and sends message m_{56} to mirror switch management job j_6 of the switch panel. Job j_7 is responsible for moving the left mirror, which receives command from j_6 via message m_{67}. Actuator job j_8 actuates the mirror movement. When job j_7 receives message m_{17} from j_1 with a higher speed value then it reacts accordingly to prohibit any mirror movement. The chosen EST and CT values are as follows: EST : $0, 20, 40$ and $65ms$; CT : $20, 20, 25$ and $20ms$ for j_5, j_6, j_7 and j_8 respectively. All jobs have to finish execution by their deadline equal to the period of $150ms$.

We consider a HW platform of 4 nodes and the node processors can have different computing power and failure rates. The slots s_0, s_1, s_2 and s_3 are statically assigned with nodes n_0, n_1, n_2 and n_3 respectively. Maximum two messages of size 25bytes can be sent from each slot. The above assumed jobs CTs are for the processors of having relative speed of 1 unit (say for $125MHz$). Let's assume the speed of node processors n_0, n_1, n_2 and n_3 are $1, 1.25(i.e, 156MHz), 1.5$ and 1.25 unit respectively. Therefore a job assigned on node n_1 requires $1/1.25$ times less time to finish the job execution. The failure rates (λ_k) per hour are $1 \times 10^{-5}, 1.5 \times 10^{-5}, 1.5 \times 10^{-5}$ and 1.75×10^{-5} for node processors n_0, n_1, n_2 and n_3 respectively.

Illustration on a Heterogeneous Platform: At this stage, mapping of above selected jobs onto available HW nodes needs to be performed. For this illustration, jobs of BFC are considered whereas jobs of both applications are considered in performing the experiments. We describe the applicability of the assignment process to a platform consists of processors of different computing power and failure rates.

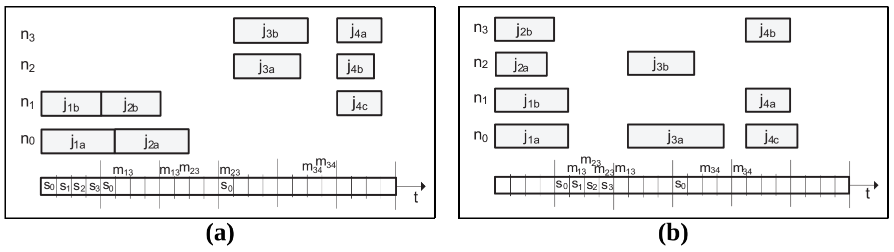


Fig. 10 Assignment of the example subsystem

We consider the assignment of jobs onto nodes shown in Figure 10. For the first mapping (Figure 10(a)) jobs are assigned without the guidance of the heuristics. For the other configuration, according to our heuristics we first assign the sensor jobs j_1 and j_2. We then apply the $HetHeus$ (described in Section 6.4.2) in $Step$ 7 of the Algorithm 2 for the remaining jobs. The resulted mapping is shown in Figure 10(b). Since all jobs are high critical only $Phase\ I$ is executed. Let us consider the case of assigning j_{3a}. We calculate the value of the product $\lambda_k \cdot T_t$ for nodes n_0, n_1, n_2 and n_3 which are $(20+25) \times 1 = 45$[7], $(16+20) \times 1.5 = 54, (14+17) \times 1.5 = 46.5$ and $(16+20) \times 1.75 = 63$ respectively. According to $HetHeus$ j_{3a} is assigned to node n_0, which has resulted the smallest product value. Similarly j_{3b} is assigned to n_2. When there is a tie, a node is chosen arbitrarily. We observe that high reliable nodes (less failure probability) execute more jobs while maintaining the scheduling length

[7] For simplicity 10^{-5} is discarded from the value of λ_k.

to a minimum. There always exists a trade-off between reliability and scheduling length. In order to maintain a better trade-off between them, this heuristic can be applied stand alone or can also be applied when there is a tie in the communication heuristic. Detailed consideration of heterogenous platform is part of our future work.

Estimation of T_N and EST: We now describe the estimation of $ESTs$ for jobs having precedence relations. In this case TDMA protocol is assumed as a communication scheme. This protocol provides deterministic access to the medium by ordering the message transmissions statically at design time and thus response time is guaranteed. Each node sends messages only during a predetermined time interval, called slot (s_i) and listens to all other nodes, over a TDMA round. In this example, the slots s_0, s_1, s_2 and s_3 of a single TDMA round are statically assigned to nodes n_0, n_1, n_2 and n_3 respectively. We assume that the slot length is equal to $4ms$ and maximum 2 messages each of size $25bytes$ can be sent per slot. For inter-job communication, it is necessary to calculate the network delay. This delay depends on the type and speed of the network. The function shown in Algorithm 1 is used for calculating T_N and actual EST. For example, job j_{3a} in Figure 10(a) can only start execution when it receives messages from job j_{1a}, j_{1b} and j_{2a}, j_{2b}. We calculate $EST_{j_{3a}}$ due to its all previous assigned jobs and select the highest one. When the delay from j_{1a} and j_{1b} are considered, j_{3a} can start executing on any nodes at $36ms$ with a T_N of $16ms$ (used Equation 5 of Section 6.1.3). However it has a precedence relation also with j_2. Therefore considering delay due to j_{2a} and j_{2b}, $EST_{j_{3a}}$ is at $52ms$ with a T_N of $12ms$. As a result $EST_{j_{3a}}$ and $EST_{j_{3b}}$ will be $52ms$. In this way actual ESTs are calculated.

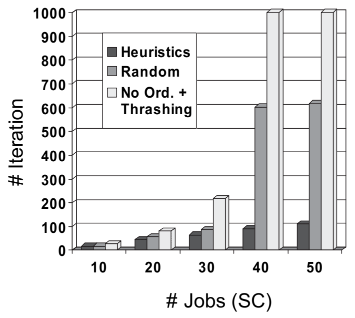
Quality Mapping: The metrics in terms of influence, total scheduling length and communication overhead have been calculated for the above mappings. The values are $(0.58, 92ms, 150bytes)$ and $(0.43, 83ms, 150bytes)$ which are corresponding to the assignment shown in Figure 10 (a) and (b) respectively. These objectives are used for measuring the quality of the mapping. The overall influence and the scheduling length have been calculated by using the formulae given in Equation 3 and 6 respectively. The communication overhead is calculated by the sum of size of the messages transferred over the network. Both mappings satisfy all the constraints, i.e, both are feasible mappings. However the mapping shown in Figure 10 (b) is preferable due to its less overall influence and scheduling length value. These three variables have also been used for our optimization framework [37], where the experimental results show a significant quantitative gain.

7.2 Performance Evaluation of the Heuristics

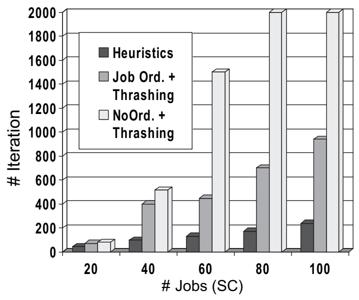
This section presents the experimental results for the proposed heuristics. Different set of jobs from 10 to 100 are randomly selected. The jobs properties are selected in the following range: Replication factor $\in 2,3,4$, $EST \in [0,80]ms$, $CT \in [2,20]ms$, $D \in [14,200]$, Memory size $\in [4,15]MB$ and Messages size $\in [2,120]Bytes$. All jobs along with their replicas are to be assigned onto a set of nodes. The memory capacities of nodes are arbitrarily chosen between $100MB$ and $250MB$. Sensors and actuators are arbitrarily attached to nodes. The proposed heuristics are compared with existing base line approaches.

The mapping problem is NP hard [13] and usually needs the guidance of heuristic technique to find a feasible solution with least iterations. If less number of iterations is needed to find the solution, it obviously takes less computation time. Our goal is to show how easily and efficiently the proposed heuristics find a feasible solution and whether the use of heuristics needs backtracking, or if backtracking is needed then how often. To show the effectiveness of the ordering in our approach the results are compared with the basic approaches where ordering heuristics are not applied. This can be effectively shown by using the number of iterations it takes to find a feasible solution. The computation time taken in each case to find the feasible solution was in the range of only few seconds. Our observation on number of iterations including number of backtracks needed to find the solution is depicted below.

Effectiveness of Heuristics: We are interested to show the performance (finding a feasible solution while reducing the complexity of the problem) of the heuristics. We observe that our multi-phase algorithm requires less or no backtracking to find a feasible solution. Several experiments are carried out. First the assignment policy is applied with the job and node ordering-heuristics, we call it *Heuristic* solution. Second, we considered random selection of nodes which is the *Random* solution. Third, we consider *Thrashing* which is a different way of exploring nodes, where first node from the order is tried for every job to be assigned. If all constraints are satisfied, the selected job is assigned onto this node, otherwise next node is explored. According to the heuristics of considering *most constrained jobs first*, high critical jobs are assigned in *Phase I* of the Algorithm 2. When jobs are assigned in this phase, different nodes are selected for them in $Step$ 6 of the algorithm. Both of these considerations result a significant number of less iterations to find the feasible solution.



(a)



(b)

Fig. 11 Performance of mapping heuristics (SC applications)

Figure 11(a) shows the number of iterations needed for different assignment policies. Five nodes are chosen for this experiment. We observe that applying the mapping heuristics takes least number of iterations and hardly need backtrack to find a feasible solution. However this does not guarantee that the backtracking is not needed at all to find feasible solutions while performing the mapping for different sets of jobs and nodes. We applied simple swapping (swap the nodes between two jobs) and reallocation (relocate a job to a different node) in the case backtrack was necessary. In case bars touch the highest iteration line (Figure 11), a feasible solution has not been found for that assignment policy despite of changing some assignments when backtrack was necessary. In Figure 11(b), the results found by heuristics process is compared with *job ordering + thrashing* and with *no job ordering + thrashing*. We observe that heuristic based solution require least number of iterations to find the feasible mapping. In this set up (Figure 11), the number of nodes were increased with the increasing number of jobs. 5 nodes are used for 20 and 40 jobs; $7,8$ and 10 nodes are for $60,80$ and 100 jobs respectively.

Figure 12 shows the outcome of the similar type of experiments, which have been performed for the non-SC jobs set. In this case the heuristics also works better than random

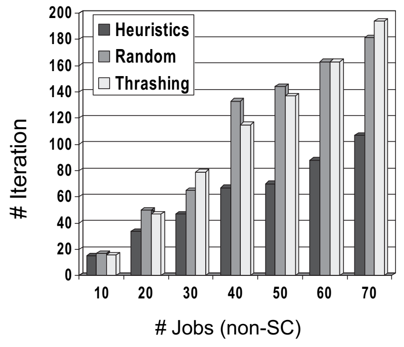


Fig. 12 Performance of mapping heuristics (non-SC applications)

or thrashing solution. As there are no high critical jobs in non-SC applications, binding constraints play role in the ordering-heuristics. Four nodes are used for this set up.

Resource Utilization: We have performed experiments in order to compare the CPU and memory utilization of heuristics process with the random and thrashing policy. We observe that the distribution of CPU and memory capacity by the heuristics approach is comparable with the random solution which is almost equally distributed among all the processors. In case of thrashing, the load (computation and memory) among nodes are not properly distributed, i.e., are not properly load balanced. The measured utilization is based on computation and memory available only for applications jobs. Resource consumption for middleware code and for other services are not included.

Observation: We have conducted similar experiments by using different application patterns. For example, deadline is set at the application level, therefore, all jobs within an application have the same deadline equal to the deadline of the application. The estimation of jobs properties are also varied, e.g., by changing the jobs criticality degree, varying the computation time, deadline, messages size etc. We observe that our heuristics approach finds similar results to those discussed above. This shows the applicability and robustness of our algorithm on a wide area of applications. Furthermore when we applied the communication heuristic, most of the communicating jobs are instantiated to allocate onto the same node. If we allow all these jobs to be assigned onto the same node, it may result poor load balancing among nodes. In order to tackle this we have applied *load balancing* technique so that the loads are properly distributed among nodes. In doing this we strive to assign the jobs equally among nodes while having the gain on communication heuristic. Load balancing is defined as the distribution of tasks among the processors such that each of the processor is loaded with almost equal amount of computation.

7.3 Validation and Comparative Study of the Approach

The result of the allocation process is validated using an independent tool [35]. The tool supports optimization-based allocation and scheduling of embedded time-triggered systems. Although, this tool uses mathematical optimization framework [36], it may lack of performance in case of large system models, so the tool and our solution proposed in this paper can complement each other.

The optimization-based tool supports MVO in the scheduling phase, including criteria such as robustness, extensibility and throughput. The developer can select an appropriate composite objective function that delivers the needed combination of the target criteria. Our method uses heuristics-based search that delivers solutions quickly. This initial solution can either be validated (proving schedulability), or optimized (to cut the solution space) using other tools. Our experiments show that the usage of allocation algorithm as part of the input to the optimization resulted in a significant performance gain. The result with less number of workloads[8] shows that in case of optimization the presence of the initial feasible solution can help to reduce the search space by at least an order of magnitude. The experiments are done using multiple objectives function as described in [35]. The independent tool is used to perform the proof-of-correctness of the output of the allocation algorithm for different size of workloads. However we experienced some limitation while performing optimization in the tool using large workloads.

Both methods are complementing each other, because *(i)* the result of the heuristic approach can be validated and scheduled using the optimization-based method, and *(ii)* the basis solution can be used to improve the performance of the optimization [37].

8 The Supporting Tool-Chain

We now develop supporting tools for the developed framework. The tool-chain provides functionality for SW-HW integration in the SC embedded systems domain. The concepts and algorithms introduced in the earlier sections have been implemented in an open, extensible development environment. First we briefly present the architecture of the integration tool-chain (Figure 13) and then describe different implementation steps of the tool-chain including transformations in VIATRA.

8.1 Implementation

The tool-chain is implemented on top of the open Eclipse tool integration framework. Eclipse is a Java-based, open-source system that is currently one of the most important open platforms for tool development [60]. As the tool-chain works with multiple modeling languages and facilitates multiple model transformations, we used VIATRA2 a generic, open-source model transformation framework [16], which is an official Eclipse extension. VIATRA2 supports the simultaneous handling of multiple models, modeling languages and transformation, as well as code generation. The model transformation is the process of converting one model to another model which applies some rules like graph transformation [61]. In this context transformation of the PIM to the PSM of the same system is performed. The input of the tool-chain are the PIM models (each containing a single application) in XMI format, the CRD model (also in XMI format), and the job code information files (in XML). In order to improve usability of the tool, we implemented a custom user interface that hides the technical details of models and transformations and shows only the high level information for the system designer. This way a VIATRA based model space is created.

After the allocation and scheduling are done, all the necessary configuration files and source code for the system can be generated. The SW is compiled and deployed using third party, system specific C compiler and SW download tools, e.g., TTP/TTX-Load [56]. All the steps shown in Figure 13 are described in the subsequent sections.

8.2 Application and platform modeling

The primary modeling artifact of the application design process (Figure **??**) is the PIM that describes three aspects of system development. The functionality contains information about jobs (basic SW components), (logical) sensors and actuators that interconnect the system with its environment, and messages that interconnect jobs. The performance (or timeliness) of the model contains requirements for message and job periods, worst-case execution

[8] Defined as numbers of jobs and nodes used in the experiment.

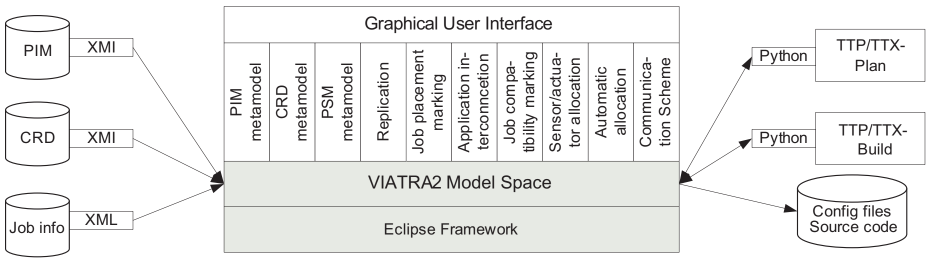


Fig. 13 Architecture of the SW-HW integration tool-chain

times, and other temporal parameters. Dependability of the PIM describes the dependability requirements such as the safety criticality degree and reliability requirements. The definition of PIM models is supported by a graphical domain-specific model editor, and a UML profile suitable for any of the current UML tools. PIMs for specific DASs are created either by using standard UML tools. Similarly to the PIM, the hardware model (called Cluster Resource Description - CRD) can also be defined using a graphical domain-specific editor.

The behavioral design step describe the definition of job behaviors. This complements the PIM by the definition of job behavior. There are several commercial tools that can be used for this purpose (like SCADE [62], or Matlab/Simulink [63]).

8.3 Marking the PIM Elements

PIM marking precedes the automatic transformations that generate the PSM of the target system. Markings are interactive steps that are used to inject additional (platform–specific) information that is not already included in PIM or in HSM. This process enables the incorporation of human design decisions into the mapping process. The marked PIM, the result of the marking process, will contain all information that is needed for automatically generating the PSM. The most important steps of marking will be introduced in the followings.

Job Placement Definition: In the *job placement definition* marking step, the designer can specify whether a job is running on one of the core nodes, or it is implemented on an external node connected to the cluster via a field bus (these jobs are called *external job*s). External allocation is usually chosen in case of hardware implementation of smart sensor/actuator nodes, like intelligent keylock modules in cars.

After the job placement definition, the designer has to manually *allocate external jobs* to field busses of the system. This allocation is currently a marking, and is not supported by automation. This decision was taken with alignment of response from industrial partners, because the allocation of external nodes to field busses has usually a predefined scheme, that cannot be changed.

Application Interconnection Definition: In this step, the designer can pair source and target gateway messages to create inter-application information flows. The interconnection information will appear explicitly in the marked PIM. Each interconnection link has an additional parameter: whether there is a need for data conversion, or not.

Sensor/Actuator Allocation: The next marking step is the *sensor/actuator allocation*. In this step, the designer has to pair logical resources (sensors/actuators) that are defined in the application PIMs with hardware peripherals of the target platform. This step is not automated, as the description of selection criteria (sensor/actuator type, accuracy, physical position in the system, etc.) would require a large effort at both the HW and SW modeling side.

Job Compatibility: This marking is the preliminary step of allocation. The designer has to define which job is compatible with which node. In case of jobs using sensors and/or actuators, it is mandatory to place them on the same node as the hardware peripheral they control (driver jobs) and the marking is done automatically. For ordinary jobs, there are no constraints on placement. The designer can, however, limit the set of possible nodes for a given job. This decision can be based on hardware constraints (presence of a specific co-processor, or peripheral), or safety criteria (in avionic application, there are physical placement constraint on nodes and SW components). The compatibility marking information is used to build the compatibility matrix in Section 6.2 needed by the mapping algorithm.

Job Code Information Import: The last marking step of the process is to import the job code information. This is a feedback from the behavioral design process, and contains information about the worst-case execution time (WCET) and code and data memory consumption of the jobs. WCET is also estimated in the PIMs, but after implementing and compiling the job code, it can be measured more precisely. The memory consumption is calculated by most compilers, and is an important input parameter for the mapping tool.

All the marking steps are implemented as Graphical User Interface (GUI) input forms. At the end of the marking process, an intermediate model (marked PIM) is built which is an input of the automatic PSM generation.

8.4 Transformations in VIATRA

Based on the input models and marking information, several automatic transformation steps are executed in order to produce the PSM of the system. These transformations are executed in the background, so the designer will see only the results and progress information and does not need to know about the transformation technology.

Replication: *Replication* of jobs and resources is the first transformation of the process. The tool replicates the high critical jobs and places them on different HW nodes to achieve higher dependability. The most important parameter of the replication is the number of replicas for a given component. This can be specified by the developer in the PIM, either on application, or on job level. It should be noted that the sensors and actuators attached to the replicated jobs are also replicated.

Job Allocation:

This transformation is the direct implementation of the allocation algorithm discussed in Section 6.4 using the capabilities of the VIATRA2 graph transformation engine. As all information required by the allocation is present in the PSM, this transformation works directly on the PSM of the system under design.

At this point, all information is available that is needed to create message and job scheduling in the cluster. We decided to use third-party scheduling tools therefore the PIM-to-PSM mapping process exports the data to the scheduler and then reads back the results. The final step of the whole process is the generation of application glue code and configuration files for the OS and for the communication controllers.

8.5 Scheduling Tool Support

A scheduling tool is necessary generate or validate the scheduling table for the time-triggered core network and for operating system instances on the core nodes. It also supports the designer in exploring several solutions in the design space and helps to design optimized implementation from RT view. Various tools exist which provide the scheduling of tasks, such as RapidRMA [64], VEST (Virginia Embedded Systems Toolkit) [65] and AIRES ToolKit [66]. These tools mainly use the RMA for uniprocessor scheduling. Nevertheless, as mentioned in the previous Section 6.1.3, TTP/TTX-Plan and TTP/TTX-Build are exploited in our approach as an example tools set for scheduling safety critical applications in a distributed platform. These tools implicitly perform a feasibility test to check whether a given system assignment is schedulable and perform the scheduling.

The schedulers are executed using input files generated by the mapping tool from the actual PSM model. The result of the scheduling is a set of configuration files for the operating system and communication subsystem of the core nodes reflecting the actual network and job shceule tables.

8.6 PSM Metamodel

The result of the PIM-to-PSM mapping is the platform specific post integration model of the system that contains all information about the SW and the HW components, and about the mapping of jobs onto nodes. The model is created and stored in the native XML format of VIATRA that is the basis of the mapping tool. The model does not need to be exported, as all transformations and marking steps are executed in the VIATRA framework.

The generated PSM controls the deployment task where all the source files (application code, middleware code, configuration code for the OS, etc.) are compiled and linked into object files and executable files. Finally, in order to run a system these files are downloaded onto the target platform , e.,g., by using the TTP/TTX-Load.

8.7 Proof of Concept Experiments

The methodologies and tool-chain introduced in this paper has been validated with various system models. The basic validation of the tools has been done using tiny examples (like the BFC and MVF introduced earlier) given by the industrial partner.

The industrial-grade experiments have been performed (using the same example) in the framework of the EU Framework 6 Integrated Project DECOS (Dependable Embedded Components and Systems) [68], where our tool-chain is used by three industrial technology demonstrators from the automotive, avionic, and industrial process control domains.

9 Conclusions

We have presented a novel transformation based integration methodology and supporting tools for the design of dependable real-time embedded systems. We performed consolidated mapping of different criticality applications onto a common computing platform, thus designing an integrated system. The functional and extra-functional requirements have been considered early in the design process, which reduces the design efforts and cost. We have presented comprehensive mapping strategies for ensuring fault-tolerance, enhancing dependability, providing schedulability analysis and have implemented them in the tool suite. Experimental results show the effectiveness, performance and robustness of our design process as it considers the following important aspects: *(i)* uses of job and node ordering-heuristics, *(ii)* checking the constraints during the assignment process (applying retrospective techniques) and are satisfied as a basis of constraints prioritization, which reduces the search space by avoiding exploring infeasible design space, *(iii)* as the algorithm uses the ordering-heuristics and run into multiple phases it takes less iterations and less convergence time, and *(iv)* the generated initial solution is used to improve the performance of the optimization. The post integration platform specific model generated by our tool-chain controls the deployment task of the system development process to run executables on the target platform. The implementation of custom user interface enhances the usability of the tool-chain providing convenient user interface, ease in importing models, one-click transformation, performing mapping algorithm, generating configuration files and results. An adaptation of the algorithm has been described considering issues such as processors having different computing power and failure rates.

References

1. Pop, P., Eles, P., Peng, Z. & Pop, T., Analysis and Optimization of Distributed Real-Time Embedded Systems, ACM Trans. Des. Autom. Electron. Syst., 11(3), 2006, 593-625.
2. Rushby, J., Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance, SRI International, NASA/CR-1999-209347, 1999.
3. Jhumka, A., Klaus, S. & Huss, S. A., A Dependability-Driven System-Level Design Approach for Embedded Systems, DATE, 2005, 372-377.
4. Sangiovanni-Vincentelli, A. & Martin, G., Platform-Based Design and Software Design Methodology for Embedded Systems, IEEE Des. Test, 18(6), 2001, 23-33.
5. Lee, Y.-H., Kim, D., Younis, M., Zhou, J. & McElroy, J., Resource Scheduling in Dependable Integrated Modular Avionics, DSN, 2000, 14-23.
6. Younis, M. F., Aboutabl, M. & Kim, D., Software Environment for Integrating Critical Real-Time Control Systems, Journal of System Architecture, 50(11), 2004, 649–674.
7. ARINC, Design Guidance for Integrated Modular Avionics, Aeronautical Radio Inc., ARINC Report 651, 1991.
8. AUTOSAR, Technical Overview V2.0.1, AUTOSAR GbR 2006.
9. Kopetz, H., Obermaisser, R., Peti, P. & N. Suri, From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems. Technical Report 22, Institut für Technische Informatik, Technische Universität Wien, Austria, Treitlstr. 1-3/182-1, 2004.
10. Peti, P., Obermaisser, R., Tagliabo, F., Marino, A. & Cerchio, S., An Integrated Architecture for Future Car Generations, ISORC, 2005, 2-13.
11. Berger, A., Embedded Systems Design: An Introduction to Processes, Tools and Techniques, CMP Books, USA, 2002.
12. OMG: Model Driven Architecture (MDA), A Technical Perspective, OMG Document No. ab/2001-02-04, Object Management Group, 2003.
13. Fernandez-Baca, D., Allocating Modules to Processors in a Distributed System, IEEE Trans. on Soft. Eng., 15(11), 1989, 1427-1436.
14. Garey, M. R. & Johnson, D. S., Computers and Intractability : A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
15. Islam, S., Lindström, R. & Suri, N., Dependability Driven Integration of Mixed Criticality SW Components, ISORC, 2006, 485-495.
16. Balogh, A. & Varró, D., Advanced Model Transformation Language Constructs in the VIATRA2 Framework, SAC, 2006, 1280-1287.
17. Ekelin, C. & Jonsson, J., Evaluation of Search Heuristics for Embedded System Scheduling Problems, Constraint Programming, 2001, 640-654.
18. Kuchcinski, K., Constraints-Driven Scheduling and Resource Assignment, ACM Trans. Des. Autom. Electron. Syst., 8(3), 2003, 355-383.
19. Wang, S., Merrick, J. R. & Shin, K. G., Component Allocation with Multiple Resource Constraints for Large Embedded Real-Time Software Design, RTAS, 2004, 219-226.
20. Rajkumar, R., Lee, C., Lehoczky, J. P. & Siewiorek, D. P., Practical Solutions for QoS-Based Resource Allocation, RTSS, 1998, 296-306.
21. Ghosh, S., Rajkumar, R., Hansen, J. & Lehoczky, J., Scalable resource allocation for multi-processor QoS Optimization, ICDCS, 2003, 174-183.
22. Kodase, S., Wang, S., Gu, Z. & Shin, K., Improving Scalability of Task Allocation and Scheduling in Large Distributed Real-Time Systems using Shared Buffers, RTAS, 2003, 181-188.
23. Oh, Y. & Son, S. H., Enhancing Fault-Tolerance in Rate-Monotonic Scheduling, Real-Time Syst., 7(3), 1994, 315–329.
24. Kandasamy, N., Hayes, J. P. & Murray, B. T., Tolerating Transient Faults in Statically Scheduled Safety-Critical Embedded Systems, SRDS, 1999, 212-221.
25. Yuan, J., Pixley, C. & Aziz, A., Constraint-Based Verification, Springer Science+Business Media, Inc., New York, USA, 2006.
26. Suri, N., Ghosh, S. & Marlowe, T., A Framework for Dependability Driven Software Integration, ICDCS, 1998, 406-415.
27. Mustafiz, S. & Kienzle, J., A Survey of Software Development Approaches Addressing Dependability, FIDJI, 2004, 78-90.
28. Effinger, M., Miller, C., Roll, W., Sharp, D. & Stuart, D., Challenges and Visions for Model-Based Integration of Avionics Systems, DASC, vol.2, 2001, 9B5/1-9B5/12.
29. Yin, X., Kiskis, D. L., Mihalik, D. & Shin, K. G., Integration of an Analysis Tool for Large-Scale Embedded Real-Time Software into a Vehicle Control Platform Development Tool Chain, ESA, 2006, 53-59.
30. Kopetz, H. & Bauer, G., The Time-Triggered Architecture, Proc. of the IEEE, 91(1), 2003, 112-126.
31. Laprie, J-C. & Randell, B., Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Trans. Dependable Secur. Comput., 1(1), 2004, 11-33.

32. Kopetz, H. & Grünsteidl, G., TTP - A Protocol for Fault-Tolerant Real-Time Systems, Computer, 27(1), 1994, 14–23.
33. The FlexRay Group, FlexRay Communications System Protocol Specification, Version 2.1, http://www.flexray.com/, 2005.
34. Rao, S., Engineering Optimization: Theory and Practice, A Wiley-Interscience Psublication, 1996.
35. Balogh, A., Pataricza, A. & Rácz, J., Scheduling of Time-Triggered Embedded Systems, EFTS, 2007, 44-49.
36. ILOG CPLEX, Optimization Tool, http://www.ilog.com/products/cplex/, 2007.
37. Islam, S. & Suri, N., A Multi Variable Optimization Approach for the Design of Integrated Dependable Real-Time Embedded Systems, EUC, LNCS - 4808/2007, 2007, 517-530.
38. Islam, S. & Omasreiter, H., Systematic Use Case Interviews for Specification of Automotive Systems, APSEC, 2005, 17-24.
39. Huber, B., Obermaisser, R. & Peti, P., MDA-Based Development in the DECOS Integrated Architecture-Modeling the Hardware Platform, ISORC, 2006, 43-52.
40. Object Management Group (OMG), Object Constraint Language 2.0 Specification, http://www.omg.org/docs/formal/06-05-01.pdf.
41. Pataricza, A., Polgár, B., Gyapay, S. & Balogh, A., Formal Checking of Metamodels and Models, DECOS/ERCIM Workshop at SAFECOMP, 2006.
42. Kandl, S., Kirner, R. & Fraser, G., Verification of Platform-Independent and Platform-Specific Semantics of Dependable Embedded Systems, WDES, 2006.
43. Kopetz, H., Real-Time Systems, Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.
44. Sadeh, N. & Fox, M. S., Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem, Artificial Intelligence, 86(1), 1996, 1-41.
45. Keichafer, R. M., Walter, C.J., Finn, A.M. & Thambidurai, P.M., The MAFT Architecture for Distributed Fault Tolerance, IEEE Trans on Comp., 37(4), 1988, 398-405.
46. Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C. & Zainlinger, R., Distributed Fault-Tolerant Real-Time Systems: The Mars Approach, IEEE Micro, 9(1), 1989, 25-40.
47. Claesson, V., Poledna, S. & Soderberg, J., The XBW Model for Dependable Real-Time Systems, ICPADS, 1998, 130-138.
48. Alstrom, K. & Torin, J., Future Architecture for Flight Control Systems, DASC, 1, 2001, 1B5/1–1B5/10.
49. Poledna, S., Barrett, P., Burns, A. & Wellings, A., Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems, IEEE Trans. on Comp., 49(2), 2000, 100-111.
50. Jhumka, A., Hiller, M. & Suri, N., Assessing Inter-Modular Error Propagation in Distributed Software, SRDS, 2001, 152-161.
51. Punnekkat, S., Burns, A. & Davis, R., Analysis of Checkpointing for Real-Time Systems, Real-Time Syst., 20(1), 2001, 83-102.
52. Izosimov, V., Pop, P., Eles, P. & Peng, Z., Design Optimization of Time-and Cost-Constrained Fault-Tolerant Distributed Embedded Systems, DATE, 2005, 864-869.
53. Ramamritham, K., Allocation and Scheduling of Precedence-Related Periodic Tasks, IEEE Trans. Parallel Distrib. Syst., 6(4), 1995, 412-420.
54. Eles, P., Peng, Z., Pop, P. & Doboli, A., Scheduling with Bus Access Optimization for Distributed Embedded Systems, IEEE Trans. Very Large Scale Integr. Syst., 8(5), 2000, 472-491.
55. Liu, J. W. S., Real-Time Systems, Prentice Hall PTR, NJ, USA, 2000.
56. TTP-Tools, TTP-Tools SW Development Suite, http://www.tttech.com/products/software.htm, 2007.
57. Silva, J. L., Metaheuristic and Multiobjective Approaches for Space Allocation, University of Nottingham, PhD thesis, 2003.
58. Rossi-Doria, O. & Paechter, B., An Hyperheuristic Approach to Course Timetabling Problem Using an Evolutionary Algorithm, Napier University, Scotland, 2003.
59. Dongarra, J., Jeannot, E., Saule, E. & Shi, Z., Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems, SPAA, 2007, 280-288.
60. Eclipse Foundation, http://www.eclipse.org/.
61. Ehrig, H., Korff, M. & Löwe, M., Tutorial Introduction to the Algebraic Approach of Graph Grammars Based on Double and Single Pushouts, Graph Grammars and Their Application to Computer Science, LNCS 532/1991, 1991, 24-37.
62. SCADE Suite, The Standard for the Development of Safety-Critical Embedded Software in the Avionics Industry, http://www.esterel-technologies.com/products/scade-suite/, 2007.
63. The MathWorks, The MathWorks Homepage, http://www.mathworks.com/, 2007.
64. RapidRMA, http://www.tripac.com/, 2004.
65. VEST, Virginia Embedded Systems Toolkit, http://www.cs.virginia.edu/∼pnn7f/vest/, 2004.
66. AIRES-ToolKit, Automatic Integration of Reusable Embedded Software, http://kabru.eecs.umich.edu/aires/, 2001.
67. Python Software Foundation, The Python Programming Language, http://www.python.org/, 2007.
68. DECOS, Dependable Embedded Components and Systems, http://www.decos.at/, IST, EU FP 6, 2004.