<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *35th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, NOV 07-10, 2016, Austin, TX.

Citation for the original published paper:

Zhang, X., Feng, L., Törngren, M., Chen, D. (2016)

Formulating Customized Specifications for Resource Allocation Problem of Distributed Embedded Systems

In: *2016 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD)* Institute of Electrical and Electronics Engineers (IEEE)

ICCAD-IEEE ACM International Conference on Computer-Aided Design

<https://doi.org/10.1145/2966986.2967042>

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-199778>

Formulating Customized Specifications for Resource Allocation Problem of Distributed Real-Time Embedded Systems

Xinhai Zhang, Lei Feng, Martin Törngren, De-Jiu Chen
(xinhai,lfeng,martint,chendj)@kth.se

KTH - Royal Institute of Technology, Stockholm, Sweden

ABSTRACT

There are plentiful attempts for increasing the efficiency, generality and optimality of the Design Space Exploration (DSE) algorithms for resource allocation problems of distributed embedded systems. Most contemporary approaches formulate DSE as an optimization or SAT problem, based on a set of predefined constraints. In this way, the end users lose the flexibility to guide and customize the exploration based on specifics of their actual problem. Besides, during the design of the DSE algorithms, manual formulation is time consuming and error-prone. To solve these problems, 1) a formal representation is defined for capturing customized architectural constraints based on a combination of propositional logic and Pseudo-Boolean (PB) formulas; 2) A process is designed to automatically translate these architectural constraints into corresponding Integer Linear Programming (ILP) constraints, commonly used for DSE. The translation process is also optimized to create ILP formulation with less introduced variables so as to reduce computation time. The results show that the generated constraints correctly reflect the corresponding specification with decent efficiency.

Keywords

DSE, Resource Allocation and Constraint Language

1. INTRODUCTION

The design of modern distributed embedded systems is becoming increasingly complex due to the increasing functional and extra-functional requirements as well as evolving platforms. Automotive embedded systems provide a relevant example domain. A key architectural design task of these systems is to deploy computational and communicational resources on a distributed hardware platform. Mehiaoui et al. [10] defined this design task in the AUTOSAR [1] framework as a PPS problem which is short for Placement (functions to execution nodes), Partitioning (functions to tasks and signals to messages) and Scheduling (assign priorities

to both tasks and messages). The complexity of this design task can be attributed to the huge amount of design options. At the same time, the final design decision should also be subject to the constraints from different perspectives such as timing, cost and performance [5].

This design task can be facilitated by Design Space Exploration (DSE) methods which automatically characterize and evaluate design alternatives and thereby suggest the feasible ones or even an optimal one [5]. There is a vast body of publications on applying DSE methods to solve this resource allocation problem. These publications try to improve the DSE methods mainly in three directions; the calculation efficiency of the exploration [17, 19, 20, 15, 18], the optimality of the suggested design decisions [19, 18] and the generality of the DSE process. Generality here can be interpreted as the realism of the assumptions, the adaptability to different use cases and the convenience for the users. As shown in Fig. 1, generality can be improved by 1) removing the assumptions that are less reasonable [19, 20, 16, 15]; 2) making more decisions so as to increase the dimensions of the design support [19, 20, 15]; and 3) considering more aspects such as security [9], reliability [2, 8] and variability [6].

Figure 1: Research Areas of improving the DSE algorithm

Another observation from most of these publications is that the architectural constraints are often pre-defined and stay fixed during the DSE process. However, this may cause some inconvenience for the users who are normally system architects. The reasons are as follows: First, most industrial problems will be slightly different and have variants of constraints; Second, new extra-functional requirements requires new constraints to be added (e.g. synchronization [16] and reliability [2]); Third, the introduction of new techniques also entails new architectural constraints (e.g. multi-core platform [12]). Meanwhile, manually formulating these new constraints is time consuming and error-prone. Therefore, there is a general need for the architectural constraints to be tailorable and customizable in a user-intelligible manner.

This paper attempts to improve the generality of the DSE methods for resource allocation problem by enabling the customizability of architectural constraints as highlighted in Fig. 1. The contributions of this paper are twofold.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

1. A formal and user-intelligible representation of architectural constraints is defined. (Section 3)
2. A process is proposed to automatically transform the architectural constraints into corresponding integer linear programming (ILP) formulation which is commonly used in this area according to our observation in literature. The transformation process is also optimized to reduce the number of introduced variables in the ILP formulation. (Section 4)

2. EXAMPLES OF ARCHITECTURAL CONSTRAINTS

Architectural constraints normally come from physical / architectural restrictions and extra-functional requirements. Observed from litterateurs, although different approaches are taken to improve DSE algorithms, similar formalisms for architectural constraints are adopted. Most of these formalisms are based on logic and inequalities. This section provides an example of a conventional constraint and an example of an ad-hoc constraint to explain this observation.

2.1 Design Variables

The elements that need to be decided are represented as variables. In these examples, only the placement of software components (SW-Cs) needs to be decided. Therefore, basic variable $x_{i,k} \in \mathbb{B}$ denotes whether software component τ_i is allocated on ECU p_k ($x_{i,k} = 1$) or not ($x_{i,k} = 0$). Here $\mathbb{B} = \{0, 1\}$ denotes the Boolean domain.

2.2 Constraints

Two examples of architectural constraints are given as follows.

EXAMPLE 1. *The schedulability of all the signals.*

This example is complex (in terms of formulation) in most of the DSE papers and is formulated slightly differently in different papers. The one used in [17, 18] is adopted here for this example. Assume that each signal is capsuled in one message. The transmissions of messages on CAN bus [3] are not preemptive, therefore a message can be blocked by not only the messages with higher priorities, but also the transmitting one with lower priority. Based on the approximation method proposed by Davis et al. [4], a sufficient but not necessary constraint can be represented as

$$(\forall s_j \in \mathcal{S}), y_{s_j} \rightarrow (WCRT_{s_j} \leq D_{s_j}) \quad (1)$$

where s_j denotes an arbitrary signal within the set of all the signals $\mathcal{S}$. D_{s_j} is the transmission deadline of s_j . y_{s_j} is an introduced variable which denotes whether s_j is transmitted on the CAN bus. Since signals are broadcasting on CAN bus. y_{s_j} can be represented as

$$y_{s_j} \leftrightarrow \bigwedge_{p \in \mathcal{P}} \neg \left(\bigwedge_{\tau_i \in \text{rec}(s_j)} x_{\tau_i, p} \wedge x_{\text{snd}(s_j), p} \right) \quad (2)$$

where p denotes an arbitrary ECU within the set of all the ECUs $\mathcal{P}$. $\text{rec}(s_j)$ denotes the set of all the receivers of s_j and $\text{snd}(s_j)$ denotes its sender.

$WCRT_{s_j}$ is the worst case response time of s_j as $WCRT_{s_j} = TT_{s_j} + Blp_{max} + Ihp_{s_j}^{ub}(\mathcal{S})$ where $TT_{s_j} = (size_{s_j} + minFr_b) / br_b$ is the transmission time of s_j . Blp_{max} is a sufficiently long

blocking time from the transmission of lower priority message. $Ihp_{s_j}^{ub}(\mathcal{S})$ refers to the interference upper bounds from all the higher priority signals. According to [4] it can be calculated as

$$Ihp_{s_j}^{ub}(\mathcal{S}) = \sum_{s_i \in hp(s_j)} y_{s_i} \times \underbrace{(D_{s_j} U_{s_i} + U_{s_i} J_{s_i} + WCTT_{s_i}(1 - U_{s_i}))}_{IhpCst_{s_j, s_i}} \quad (3)$$

where $WCTT_{s_i}$, D_{s_i} and J_{s_i} respectively denote the worst case transmission time, the deadline and the jitter of s_i . $U_{s_j} = WCTT_{s_i} / T_{s_i}$ with T_{s_i} is the period of s_i . $hp(s_j)$ returns the set of all signals that has higher priority than s_j . Given s_i and s_j , $IhpCst_{s_j, s_i}$ is a constant value.

EXAMPLE 2. *Some OEM may define a particular ECU to have CPU overload if its CPU utilization is over 70% and memory overload if its memory utilization is over 80%. The OEM may have an ad hoc specification for this system that 1) there are at most one ECU that has both CPU and memory overload; and 2) there are at most two ECUs with either CPU overload or memory overload.*

This example can be represented as

$$\begin{cases} \sum_{p_k \in \mathcal{P}} (\text{CPU_overload}(p_k) \wedge \text{Mem_overload}(p_k)) \leq 1 \\ \sum_{p_k \in \mathcal{P}} (\text{CPU_overload}(p_k) \vee \text{Mem_overload}(p_k)) \leq 2 \end{cases} \quad (4)$$

where

$$\begin{cases} \text{CPU_overload}(p_k) \leftrightarrow (\sum_{\tau_i \in \mathcal{T}} (U_{\tau_i}) x_{i,k} \leq 0.7) \\ \text{Mem_overload}(p_k) \leftrightarrow (\sum_{\tau_i \in \mathcal{T}} m_{\tau_i} x_{i,k} \leq mt_{p_k} * 0.8) \end{cases}$$

where $U_{\tau_i} = WCET_{\tau_i} / T_{\tau_i}$ is the CPU utilization of SW-C τ_i . m_{τ_i} is the memory utilization of τ_i . mt_{p_k} is the total memory of ECU p_k .

3. FORMAL REPRESENTATION OF ARCHITECTURAL CONSTRAINTS

In order to automatically formulate all the customized constraints, they firstly have to be represented in a formal manner so as to be processable. As analyzed in the previous sections, the formal representation should combine proposition logic and inequalities. Even though ILOG CPLEX¹ [7], which is a commonly used optimization solver for DSE for embedded system architecture, supports logical construction of linear constraints when solving ILP problem, it is difficult to express arbitrary compound logical propositions due to the restriction from the CPLEX syntax like 1) Boolean type cannot be used with decision expression and 2) parentheses are not supported within logical expressions to specify the order in which logical operators in a compound proposition are to be applied. The rest of this section introduces our method to formally describe architectural constraints with arbitrary combinations of linear **Pseudo Boolean (PB)** functions [11] and propositional logic. The definitions presented in this section are used in the rest of the paper.

DEFINITION 1. *A linear PB function $f(\mathbf{x})$ is defined as*

$$f : \mathbb{B}^n \rightarrow \mathbb{R} \\ f(\mathbf{x}) = \mathbf{a}^T \cdot \mathbf{x} = \sum_{i=1}^n a_i x_i \quad x_i \in \mathbb{B}, a_i \in \mathbb{R} \quad (5)$$

¹By the time when the paper is written, the latest version of CPLEX is V12.6.2.

where $n \in \mathbb{Z}^+$ is the arity of the function.

DEFINITION 2. An **atomic PB proposition** $\alpha(\mathbf{x})$ is a function that compares a PB function $f(\mathbf{x})$ with a real number m . It is defined as

$$\alpha : \mathbb{B}^n \rightarrow \mathbf{B} \quad \alpha(\mathbf{x}) = (f(\mathbf{x}) \text{ op } m) \quad (6)$$

where $\text{op} \in \{<, \leq, =, \neq, \geq, >\}$, $m \in \mathbb{R}$

op is a comparison operator that used in equations and inequalities; $\mathbf{B} = \{\text{true}, \text{false}\}$ is the domain of truth value.

It means that the atomic PB proposition $\alpha(\mathbf{x}) = \text{true}$ iff the corresponding PB inequality/equation $f(\mathbf{x}) \text{ op } m$ is satisfied, otherwise $\alpha(\mathbf{x}) = \text{false}$. There are two special kinds of proposition namely $x_i = 1$ and $x_i = 0$. They only represent the logical meaning of the variables. To simplify in the further text, proposition $x_i = 1$ is represented as x_i and $x_i = 0$ is represented as $\neg x_i$. To distinguish with other types of PB proposition, x_i and $\neg x_i$ are called **pseudo logic variables**. In the rest of the paper, the term atomic PB proposition does not refer to pseudo logic variables.

DEFINITION 3. A **compound PB proposition (CPBP)** $p : \mathbb{B}^n \rightarrow \mathbf{B}$ is a function that combines a set of atomic PB propositions and pseudo logic variables with propositional logic operators.

Propositional logic operators include $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (conditional statement) and $\leftrightarrow$ (biconditional). Formulae (2) and (4) are examples of compound PB propositions. In this paper, based on the observation from literature and the examples given in Section 2, the following assumption is proposed.

ASSUMPTION 1. Most of the architectural constraints regarding embedded system architecture can be described by one or several CPBPs.

4. AUTOMATIC TRANSFORMATION

The overall idea to transform these CPBP constraints is to firstly convert the constraints into a normal form which is uniform and easier for the formulator to process. Secondly, a formulator is designed to transform the constraints from the normal form to ILP constraints. Therefore, Section 4.1 defines the normal forms and describes how to transform from arbitrary forms into normal forms; Section 4.2 introduces how a normal-formed constraint is automatically transformed into ILP constraints; finally, in Section 4.3 the formulation process is optimized in terms of the efficiency of the generated formulation.

4.1 Normal Forms

This section defines the normal forms for PB propositions (7) and CPBP (12). Moreover, the transformation methods from arbitrary forms to normal forms are also described and proved in this section.

4.1.1 Normal Form for atomic PB Propositions

The normal form of an atomic PB Proposition $a(\mathbf{x})$ is defined as²

$$\alpha(\mathbf{x}) \Leftrightarrow \mathbf{a}^T \cdot \mathbf{x} \leq m \quad m \in \mathbb{R} \quad (7)$$

² Symbol $\Leftrightarrow$ denotes the logical equality of the two propositions on both sides while $\leftrightarrow$ is just a logic operator within a proposition. They are logically the same, but two different representations in different cases to clarify the description.

According to (6), there are five other forms of atomic PB propositions according to the type of the comparison operator op . All the atomic PB propositions in other forms can be converted into the normal form or a compound PB proposition whose clauses are all in the normal form. The conversion method is introduced as follows.

The conversion for $\mathbf{a}^T \cdot \mathbf{x} < m$ is firstly introduced. For a fixed $\mathbf{a} \in \mathbb{R}^n$, let A be the set of all the possible values of $\mathbf{a}^T \cdot \mathbf{x}$. Since $\mathbf{x} \in \mathbb{B}^n$, therefore A is discrete and finite. If $m \notin A$, $\mathbf{a}^T \cdot \mathbf{x} < m$ is logically equivalent to $\mathbf{a}^T \cdot \mathbf{x} \leq m$, since $\mathbf{a}^T \cdot \mathbf{x} = m$ will never be satisfied. Problem arises when $m \in A$. The following theorem is proposed to solve this problem.

THEOREM 1. There exists a positive Δ within the range $(0, \Delta_{\max}]$ that

$$(\mathbf{a}^T \cdot \mathbf{x} < m) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \leq m - \Delta) \quad (8)$$

with $\Delta_{\max} =$

$$\begin{cases} m - \max\{\mathbf{a}^T \cdot \mathbf{x} \mid \mathbf{a}^T \cdot \mathbf{x} < m\} & \text{if } \exists \mathbf{x}, \mathbf{a}^T \cdot \mathbf{x} < m \\ \text{any positive value} & \text{otherwise} \end{cases} \quad (9)$$

Limited by the space, the proof is not represented in the paper. In the case where $\mathbf{a} \in \mathbb{Z}^n$, the value of Δ can be easily determined by observation without solving the optimization problem (9). In this case

$$\Delta = \begin{cases} m - \lfloor m \rfloor & \text{if } m \notin \mathbb{Z} \\ 1 & \text{if } m \in \mathbb{Z} \end{cases}$$

Otherwise, no direct way has been found to determine the value of Δ without calculating the optimization problem (9) which is actually an linear programming problem. In order to avoid solving the optimization in all the cases, an alternative and generic solution is given based on the following assumption to the parameters in the architecture model.

ASSUMPTION 2. All the data in the architectural model is measured data or estimated data, therefore it is accurate to a certain decimal place. It means that all the data in the architectural model has only specific decimal places.

This can be described as

$$\mathbf{a} \in \mathbb{R}_p^n \quad \text{where} \\ \mathbb{R}_p \subset \mathbb{R} \quad \text{with } a_i \in \mathbb{R}_p \text{ iff } \lfloor a_i \times 10^p \rfloor = a_i \times 10^p$$

where p is the number of decimal places that all the data has. In this setting, the following theorem is given.

THEOREM 2. The proposition (8) is always true if $\Delta = 10^{-p}$ with $\mathbf{a} \in (\mathbb{R}_p)^n$ and $m \in \mathbb{R}_p$.

The case where $\mathbf{a} \in \mathbb{Z}^n$ and $m \in \mathbb{Z}$ is special as $\mathbb{Z} = \mathbb{R}_0$. Based on Assumption 2 and Theorem 2, in the rest of the paper $\Delta = 10^{-p}$ where p is specified by the architecture model.

Even though the PB proposition $\alpha_{eq}(\mathbf{x}) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} = m)$ is a linear constraint for ILP, it can not be directly formulated when it is compounded with logical relations. Therefore, it also needs to be transformed into a compound PB proposition:

$$\alpha_{eq}(\mathbf{x}) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \leq m) \wedge (\neg \mathbf{a}^T \cdot \mathbf{x} \leq -m) \quad (10)$$

Note that pseudo logic variables x_i and $\neg x_i$ are exceptions. Even though they are actually PB propositions as $x_i = 1$ $x_i = 0$, they are treated as individual logic variables. Therefore they do not need to be transformed into the above normal form.

The PB proposition $\alpha_{neq}(\mathbf{x}) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \neq m)$ can be transformed as

$$\alpha_{neq} \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \leq m - \Delta) \vee (-\mathbf{a}^T \cdot \mathbf{x} \leq -m - \Delta) \quad (11)$$

Simply, the PB proposition $\alpha_{geq}(\mathbf{x}) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \geq m)$ can be transformed into $-\mathbf{a}^T \cdot \mathbf{x} \leq -m$; and the PB proposition $\alpha_{gt}(\mathbf{x}) \Leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} > m)$ can be transformed into $-\mathbf{a}^T \cdot \mathbf{x} \leq -m - \Delta$ or $\neg(\mathbf{a}^T \cdot \mathbf{x} \leq m)$.

4.1.2 Normal Form for CPBPs

According to Assumption 1, CPBP is the general form for all the customizable constraints. The normal form for CPBPs is a conjunctive normal form (CNF) [13] of atomic PB propositions and pseudo logic variables. It has been proved that all the propositional logic can be converted into CNF [13] and solvers exist to provide automatic conversion such as PBL package³. Even though the CNF conversion is an important step during the automatic formulation, it is not the emphasis of this paper. Some of the CNF convertor in some cases can lead to exponential blowup in size. However the input size of the CNF convertor is the number of clauses of the input CPBP, but not the total number of all the design variables. Each atomic PB proposition is only counted as one clause. From our observation, normally, most of the constraints only contains a limited number of clauses. Therefore, The following assumption is made to the CNF conversion process.

ASSUMPTION 3. *The CNF conversion is performed by a third-party CNF convertor whereby all the customized CPBP constraints can be converted into CNF within reasonable size and time.*

According to the definition of CNF, each clause of a CNF only contains two types of logic operators which are OR ($\vee$) and NEGATION ($\neg$). According to the associative law for disjunction, the normal form for a CPBP p_i is:

$$p_i \Leftrightarrow \bigwedge_{c_j \in C_{p_i}} \left(\left(\bigvee_{x \in L_{c_j}} x \right) \vee \left(\bigvee_{x \in Ln_{c_j}} \neg x \right) \vee \left(\bigvee_{\alpha(\mathbf{x}) \in A_{c_j}} \alpha(\mathbf{x}) \right) \vee \left(\bigvee_{\alpha(\mathbf{x}) \in An_{c_j}} \neg \alpha(\mathbf{x}) \right) \right) \quad (12)$$

where C_{p_i} is the set of all the clauses of the CNF. In each clause, L_{c_j} , Ln_{c_j} , A_{c_j} and An_{c_j} denote the sets of four kinds of propositions that C_{p_i} may have, namely pseudo logic variables x and $\neg x$; and PB propositions $\alpha(\mathbf{x})$ and $\neg \alpha(\mathbf{x})$. All the PB propositions $\alpha(\mathbf{x})$ are in their normal form as (7).

4.2 Normal Form to ILP Constraints

This section introduces how to translate a customized constraint represented as (12) into a set of constraints that are acceptable for generic binary programming solvers or ILP

solvers. These solvers normally accept constraints in the following two types.

$$\mathbf{a}^T \cdot \mathbf{x} \leq al \quad \text{OR} \quad \mathbf{b}^T \cdot \mathbf{x} = beq$$

Since the relationship between the clauses of (12) is conjunction, all the clauses need to be satisfied to guarantee the overall satisfaction. Therefore, constraints in form (12) can be broken down into a set of its clauses.

Each CNF clause can be directly handle by ILOG OPL as logical constraint. The rest of this paper introduces our method to transform one CNF clause into generic ILP constraints. Our method is also compared with ILOG OPL's inherent method to handle logical constraints in terms of efficiency in Section 5. Our formulation contains the following cases.

Case 1: If a CNF clause c_j is only an atomic PB proposition (as $L_{c_j} = Ln_{c_j} = \emptyset$ and $|A_{c_j}| + |An_{c_j}| = 1$), this proposition can be directly used as one ILP constraint. If the proposition is $\neg \alpha(\mathbf{x})$, it can be converted into its normal form according to (8) as

$$\neg(\mathbf{a}^T \cdot \mathbf{x} \leq m) \Leftrightarrow (-\mathbf{a}^T \cdot \mathbf{x} \leq -m + \Delta) \quad (13)$$

Case 2: If a CNF clause c_j contains no PB propositions (as $A_{c_j} = An_{c_j} = \emptyset$), c_j can be formulated as

$$-\sum_{x_i \in L_{c_j}} x_i - \sum_{x_i \in Ln_{c_j}} (1 - x_i) \leq -1$$

because as long as there exist one $x_i \in L_{c_j}$ that $x_i = 1$ or there exist one $x_i \in Ln_{c_j}$ that $x_i = 0$, both the clause c_j and its ILP formulation are satisfied. Otherwise, both of them are evaluated as false.

Case 3: In this case, a CNF clause c_j has $L_{c_j} \neq \emptyset$, $Ln_{c_j} \neq \emptyset$ and $|A_{c_j}| + |An_{c_j}| = 1$. After converting $\neg \alpha(\mathbf{x})$ into its normal form according to (13), it can be represented as

$$\left(\bigvee_{x \in L_{c_j}} x \right) \vee \left(\bigvee_{x \in Ln_{c_j}} \neg x \right) \vee (\mathbf{a}^T \cdot \mathbf{x} \leq m) \quad (14)$$

THEOREM 3. *A CNF clause in form of (14) is transformed as*

$$-\mathcal{M} \left(\sum_{x_i \in L_{c_j}} x_i + \sum_{x_i \in Ln_{c_j}} (1 - x_i) \right) + \mathbf{a}^T \cdot \mathbf{x} \leq m \quad (15)$$

where $\mathcal{M} \geq \mathcal{M}_{min}$ and

$$\mathcal{M}_{min} = \begin{cases} \max(\mathbf{a}^T \cdot \mathbf{x}) - m & \text{if } \max(\mathbf{a}^T \cdot \mathbf{x}) \geq m \\ 0 & \text{if } \max(\mathbf{a}^T \cdot \mathbf{x}) < m \end{cases}$$

where

$$\max(\mathbf{a}^T \cdot \mathbf{x}) = \sum b_i \quad \text{with} \quad b_i = \begin{cases} a_i & \text{if } a_i \geq 0 \\ 0 & \text{if } a_i < 0 \end{cases}$$

$\mathcal{M}$ is a big enough positive number to guarantee that as long as the formula in the following bracket is greater than or equal to one the inequality (15) should always hold. The proof of this theorem is elaborated in appendix.

PROOF. For simplicity, let

$$s(\mathbf{x}) = \left(\sum_{x_i \in L_{c_j}} x_i + \sum_{x_i \in Ln_{c_j}} (1 - x_i) \right)$$

³PBL package web interface <http://formal.cs.utah.edu:8080/pbl/PBL.php>

Proposition (14) can be interpreted as

$$\begin{cases} \exists x_i \in L_{c_j} & x_i = 1 & \Rightarrow & c_j \text{ holds} \\ \exists x_i \in Ln_{c_j} & x_i = 0 & \Rightarrow & c_j \text{ holds} \\ \mathbf{a}^T \cdot \mathbf{x} \leq m & & \Rightarrow & c_j \text{ holds} \\ \text{otherwise} & & \Rightarrow & c_j \text{ is false} \end{cases}$$

The equivalence between (14) and (15) is proved by verifying all these four conditions on (15).

In the first and second conditions

$$((\exists x_i \in L_{c_j}, x_i = 1) \text{ OR } (\exists x_i \in Ln_{c_j}, x_i = 0)) \Rightarrow s(\mathbf{x}) \geq 1$$

When $\max(\mathbf{a}^T \cdot \mathbf{x}) < m$, since $\mathcal{M}$ is positive, $-\mathcal{M}s(\mathbf{x})$ is always less than 0. Therefore

$$-\mathcal{M}s(\mathbf{x}) + \mathbf{a}^T \cdot \mathbf{x} < \mathbf{a}^T \cdot \mathbf{x} \leq m$$

Thus, both (14) and the MILP constraint (15) always hold.

When $\max(\mathbf{a}^T \cdot \mathbf{x}) \geq m$, because $\mathcal{M} \geq \max(\mathbf{a}^T \cdot \mathbf{x}) - m$ and $\mathcal{M} > 0$, thus

$$-\mathcal{M}s(x) < -\mathcal{M} \leq -(\max(\mathbf{a}^T \cdot \mathbf{x}) - m)$$

Both (14) and (15) are also always satisfied. Therefore the equivalence between (14) and (15) in the first two conditions is proved.

In the third condition, $-\mathcal{M}s(x) \leq 0$ and also $\mathbf{a}^T \cdot \mathbf{x} \leq m$. The proof can be simply archived by summing up these two inequalities.

In the fourth condition, $-\mathcal{M}s(x) = 0$ and $\mathbf{a}^T \cdot \mathbf{x} > m$. Therefore, neither (14) or the MILP constraint (15) is never satisfied in this condition. $\square$

Case 4: This is the generic case with $|A_{c_j}| + |An_{c_j}| > 1$. Unfortunately, no elegant method has yet been found to handle this case directly. It has to be transformed to Case 3 by replacing extra propositions with intermediate helper variables $\mathbf{z}$ along with extra constraints. For example, the CNF clause

$$(x_1 \vee x_2) \vee (\neg x_1 \vee \neg x_3) \vee (\mathbf{a}_1^T \cdot \mathbf{x} \leq m_1) \vee (\mathbf{a}_2^T \cdot \mathbf{x} \leq m_2) \vee \neg(\mathbf{a}_3^T \cdot \mathbf{x} \leq m_3)$$

can be transformed as follows to fit Case 2.

$$(x_1 \vee x_2 \vee z_1) \vee (\neg x_1 \vee \neg x_3 \vee \neg z_2) \vee (\mathbf{a}_1^T \cdot \mathbf{x} \leq m_1) \\ \text{with } \begin{cases} z_1 \leftrightarrow ((\mathbf{a}_2^T \cdot \mathbf{x} \leq m_2)) \\ z_2 \leftrightarrow ((\mathbf{a}_3^T \cdot \mathbf{x} \leq m_3)) \end{cases}$$

Intermediate variables z_1 and z_2 are introduced to replace the extra atomic PB propositions ($\mathbf{a}_2^T \cdot \mathbf{x} \leq m_2$ and $\mathbf{a}_3^T \cdot \mathbf{x} \leq m_3$). The relation between introduced variables and the replaced atomic PB propositions is restricted by the newly introduced CPBP constraints. In this way, the main proposition can be formulated according to Theorem 3 as

$$-\mathcal{M}(x_1 + x_2 + z_1 + (1 - x_1) + (1 - x_3) + (1 - z_2)) + \mathbf{a}_1^T \cdot \mathbf{x} \leq m_1$$

The CPBP constraint for the intermediate variables can be genetically represented as

$$z_i \leftrightarrow (\mathbf{a}^T \cdot \mathbf{x} \leq m)$$

To transform it into ILP constraints, it is firstly transformed into CNF form as

$$(\neg z_i \vee (\mathbf{a}^T \cdot \mathbf{x} \leq m)) \wedge (z_i \vee (\neg \mathbf{a}^T \cdot \mathbf{x} \leq -m - \Delta))$$

which can be further transformed into ILP constraints according to Theorem 3 as

$$\begin{cases} -\mathcal{M}_1(1 - z_i) + \mathbf{a}^T \cdot \mathbf{x} \leq m \\ -\mathcal{M}_2 z_i - \mathbf{a}^T \cdot \mathbf{x} \leq -m - \Delta \end{cases} \quad (16)$$

4.3 Optimized Formulation Process

The result of the formulation will be given to an ILP solver. Since the software allocation problem is NP hard [14], the efficiency of the formulation has to be considered. Regarding the efficiency of solving ILP, we claim that **increasing the number of variables significantly increases the computation time of ILP solvers**, since it extends the design space. Therefore, three optimization methods (OMs) are proposed to reduce the number of variables in the generated formulation so as to improve the efficiency.

OM1: As mentioned in Section 4.1, pseudo logic variable $x_i = 1$ $x_i = 0$ are exceptions of PB propositions. They do not need to be transformed into the normal form as given in (10). Another kind of exceptions are the propositions in form $x_i = x_j$ or $x_i \neq x_j$. These propositions can be transformed into their normal forms. However doing so increases the number of atomic PB propositions which may further increase the number of introduced variables z . Another way to process this kind of propositions is

$$(x_i = x_j) \Leftrightarrow (x_i \leftrightarrow x_j) \Leftrightarrow ((\neg x_i \vee x_j) \wedge (\neg x_j \vee x_i)) \\ (x_i \neq x_j) \Leftrightarrow (x_i \leftrightarrow \neg x_j) \Leftrightarrow ((\neg x_i \vee \neg x_j) \wedge (x_j \vee x_i))$$

In this way, only the number of pseudo logic variables in each CNF clauses is increased. According to the formulation from CPBP to ILP constraints, increasing the number of pseudo logic variables needs not to introduce new variables. Therefore, this kind of transformation is better than transforming them into the normal form.

OM2: It is commonly seen that some constraints are used to define an introduced variable. These kind of constraints are called definition PB proposition.

DEFINITION 4. A *definition PB proposition* $p(\mathbf{x})$ is a CPBP that is in form $x_i \leftrightarrow p_{x_i}(\mathbf{x})$. $p_{x_i}(\mathbf{x})$ is a CPBP that other than $p(\mathbf{x})$. It is called the definition of x_i .

If in all the CPBP constraints, a defined variable x_i does not exist in any PB functions, all the x_i can be directly replaced by its definition $p_{x_i}(\mathbf{x})$. In other words, if x_i only exists as pseudo logic variable in all the constraints, it can be replaced by its definition. For example

$$\begin{cases} (x_1 + x_2 + x_3 \leq 2) \wedge y \vee x_3 \\ y \leftrightarrow (x_2 \vee (2.4x_1 - 0.7x_2 + 3.3x_3 \leq 2.6)) \end{cases}$$

can be optimized as

$$(x_1 + x_2 + x_3 \leq 2) \wedge (x_2 \vee (2.4x_1 - 0.7x_2 + 3.3x_3 \leq 2.6)) \vee x_3$$

OM3: As mentioned, if a CNF clause contains more than one atomic PB propositions, extra propositions need to be replaced by new variables. In some cases, the choice of which proposition to replace may also affect the number of new variables. For example the following constraints need to be formulated.

$$\begin{cases} \dots \vee \alpha_1(\mathbf{x}) \vee \alpha_2(\mathbf{x}) \\ \dots \vee \alpha_2(\mathbf{x}) \vee \neg \alpha_3(\mathbf{x}) \\ \dots \end{cases}$$

The first constraint contains two atomic PB propositions. It implies that either $\alpha_1(\mathbf{x})$ or $\alpha_2(\mathbf{x})$ needs to be replaced by new variable. If $\alpha_1(\mathbf{x})$ is replaced with new variable z_1 , when it comes to the second constraint, another new variable still needs to be introduced to replace either $\alpha_2(\mathbf{x})$ or $\alpha_3(\mathbf{x})$. However, if $\alpha_2(\mathbf{x})$ is chosen to be replaced with new variable when processing the first constraint, no new variable needs to be introduced to process the second constraint since it has only one atomic PB proposition left.

Figure 2: Optimized formulation process

In real situation, the same atomic PB proposition may appears in different clauses of one constraint or even appears in different CPBP constraints. Therefore, an algorithm is needed to globally process the CNF clauses that contains multiple atomic PB propositions. As shown in Fig. 2, this OM should be performed after all the CPBP constraints are transformed into CNF format.

Assume that there are in total m CNF clauses (as $c_1, c_2, \dots, c_m$) that have more than one atomic PB propositions and l atomic PB propositions (as $\alpha_1, \alpha_2, \dots, \alpha_l$) that appear n times in the m clauses. Let n_{α_i} denote the occurrence time of proposition α_i in the m clauses, then $n = \sum_{i=1}^l n_{\alpha_i}$ and $n \geq 2m$. The goal of OM3 is to replace as less α as possible and also to guarantee that each clause retains no more than one atomic PB proposition. Seen from Fig. 2, OM3 is part of a recursion. Since one proposition can appear only once in one CNF clause, therefore, the OM3 is design as shown in Algorithm 1. In each recursion, it only replace the proposition with the most occurrences. After OM3, the formulator will re-find the clauses that still has multiple atomic PB propositions and re-execute OM3 to these clauses until no such clause can be found.

5. VERIFICATION

The automatic ILP constraints generation method is verified on the two examples illustrated in Section 2. The correctness and the efficiency of the generated formulae are verified by comparing with traditional formulation methods and the internal procedures of ILOG OPL to handle logical constraints.

5.1 Example 1

Assuming the signal s_1 has two receivers as $rec(s_1) = \{\tau_2, \tau_3\}$ and $snd(s_1) = \tau_1$. τ_1, τ_2 and τ_3 can be freely placed on 3 ECUs p_1, p_2 and p_3 which are connected on one CAN

Algorithm 1 Global process ($\mathcal{C}_{all}$)

- 1: Let the set of all the under processing CNF clauses (the ones with multiple atomic PB propositions) $\mathcal{C}_{up} := \emptyset$
- 2: **for** each CNF clause $c_i \in \mathcal{C}_{all}$ **do**
- 3: **if** c_i has multiple atomic PB propositions **then**
- 4: $\mathcal{C}_{up} := \mathcal{C}_{up} \cup \{c_i\}$
- 5: **for** each atomic PB propositions α_j of c_i **do**
- 6: Log the total occurrence time of α_j
- 7: Log c_i as one affiliated clause of α_j
- 8: Let $\alpha_{most} :=$ the atomic PB proposition with the most occurrence
- 9: Let $\mathcal{C}_{\alpha_{most}} :=$ the set of all the logged clauses that contain α_{most}
- 10: **for** each CNF clause $c_i \in \mathcal{C}_{\alpha_{most}}$ **do**
- 11: Replace α_{most} of c_i with new variable $z_{\alpha_{most}}$
- 12: Add new constraints for $z_{\alpha_{most}}$

bus b . According to the formulation method described in section 4, proposition (1) is transformed into CNF as

$$\neg y_{s_1} \vee (TT_{s_1} + Blp_{max} + \sum_{s_i \in hp(s_1)} IhpCst_{s_1, s_i} y_{s_i} \leq D_{s_1})$$

which can be further transformed into ILP constraints as

$$-\mathcal{M}(1 - y_{s_1}) + TT_{s_1} + Blp_{max} + \sum_{s_i \in hp(s_1)} IhpCst_{s_1, s_i} y_{s_i} \leq D_{s_1}$$

$$\text{with } \mathcal{M} = \sum_{s_i \in hp(s_1)} IhpCst_{s_1, s_i}$$

Since $rec(s_1) = \tau_2$ and $snd(s_1) = \tau_1$. The CNF form of proposition (3) is

$$\begin{aligned} &(\neg y_{s_1}, \neg x_{1,1}, \neg x_{2,1}, \neg x_{3,1})(\neg y_{s_1}, \neg x_{1,2}, \neg x_{2,2}, \neg x_{3,2}) \\ &(\neg y_{s_1}, \neg x_{1,3}, \neg x_{2,3}, \neg x_{3,3}) \\ &(x_{1,1}, x_{1,2}, x_{1,3}, y_{s_1})(x_{1,1}, x_{1,2}, x_{2,3}, y_{s_1})(x_{1,1}, x_{1,2}, x_{3,3}, y_{s_1}) \\ &(x_{1,1}, x_{2,2}, x_{1,3}, y_{s_1})(x_{1,1}, x_{2,2}, x_{2,3}, y_{s_1})(x_{1,1}, x_{2,2}, x_{3,3}, y_{s_1}) \\ &(x_{1,1}, x_{3,2}, x_{1,3}, y_{s_1})(x_{1,1}, x_{3,2}, x_{2,3}, y_{s_1})(x_{1,1}, x_{3,2}, x_{3,3}, y_{s_1}) \\ &(x_{2,1}, x_{1,2}, x_{1,3}, y_{s_1})(x_{2,1}, x_{1,2}, x_{2,3}, y_{s_1})(x_{2,1}, x_{1,2}, x_{3,3}, y_{s_1}) \\ &(x_{2,1}, x_{2,2}, x_{1,3}, y_{s_1})(x_{2,1}, x_{2,2}, x_{2,3}, y_{s_1})(x_{2,1}, x_{2,2}, x_{3,3}, y_{s_1}) \\ &(x_{2,1}, x_{3,2}, x_{1,3}, y_{s_1})(x_{2,1}, x_{3,2}, x_{2,3}, y_{s_1})(x_{2,1}, x_{3,2}, x_{3,3}, y_{s_1}) \\ &(x_{3,1}, x_{1,2}, x_{1,3}, y_{s_1})(x_{3,1}, x_{1,2}, x_{2,3}, y_{s_1})(x_{3,1}, x_{1,2}, x_{3,3}, y_{s_1}) \\ &(x_{3,1}, x_{2,2}, x_{1,3}, y_{s_1})(x_{3,1}, x_{2,2}, x_{2,3}, y_{s_1})(x_{3,1}, x_{2,2}, x_{3,3}, y_{s_1}) \\ &(x_{3,1}, x_{3,2}, x_{1,3}, y_{s_1})(x_{3,1}, x_{3,2}, x_{2,3}, y_{s_1})(x_{3,1}, x_{3,2}, x_{3,3}, y_{s_1}) \end{aligned}$$

where all the $\wedge$ between each CNF clauses are removed and all the $\vee$ within each CNF clause are replaced by "," to save space. This CNF is further transformed into ILP constraints by the method proposed in this paper. Proposition (1) is linearized in [17, 18] as

$$\begin{aligned} y_{s_1} &= 1 - \sum_{p_k \in \mathcal{P}} \delta_{s_1, p_k} \quad \text{with } \forall p_k \in \mathcal{P} \\ \begin{cases} x_{snd(s_1), k} + x_{rec(s_1), k} \geq 2 * \delta_{s_1, p_k} \\ x_{snd(s_1), k} + x_{rec(s_1), k} - \delta_{s_1, p_k} \leq 1 \end{cases} \end{aligned} \quad (17)$$

Even though Zhu et al. [20] and Wozniak et al. [15] use different schedulability model, they linearize y_{s_j} in the same way. Comparing with their method, the constraints generated by our method saves a big amount of variables δ .

The generated ILP constraints by our method and the traditional formulation (17) were compared in the same optimization environment. The design cases used in this environment only need to decide the allocation of SW-Cs to ECUs. It minimizes CAN utilization subject to the constraints from memory utilization, schedulability of all the SW-Cs, and the schedulability of all the signals. Except the constraints for the schedulability of all the signals, all other constraints are identical. The optimization is performed in CPLEX on the same computer which has dual-core CPU @ 2.6GHz and 8 GB memory. The comparison is done on four cases with different sizes of system architecture. The size of system architecture is characterized by the number of SW-Cs, ECUs and signals as $size = (|\mathcal{T}|, |\mathcal{P}|, |\mathcal{S}|)$. The result of the comparison is illustrated in Table 1 with F1 denoting the traditional formulation, F2 denoting our approach and FSs denoting the feasible solutions found during the exploration.

Table 1: Experiment result of example 1

Idx		Size	Objective Value	No. of FSs	Time in Snd.
#1	F1	(48,5,82)	48	21	52
	F2		48	10	3
#2	F1	(48,6,82)	52.6	25	227
	F2		52.6	25	10
#3	F1	(48,7,82)	28.5	28	417
	F2		28.5	20	16
#4	F1	(48,7,92)	36.4	23	434
	F2		36.4	26	29

The results show that the signal schedulability constraints generated from our method produce the same result as the traditional formulation method. Obviously, exploration with our method takes much less computation time. However, the CNF of y_{s_j} in proposition (1) has exponential size with the number of ECUs in the system architecture. Therefore, this kind of formulation can only be used for small sized system according to Assumption 3. But if this kind of formulation is used for a set of ECUs and traditional formulation is used for the other ECUs, the computation time can still be largely reduced. Therefore, if our method can be co-designed with the CNF convertor and the ILP solver, a particular size of the set can be determined. In this way, an optimal trade-off can be made between efficiency and feasibility.

5.2 Example 2

Example 2 describes an ad-hoc specification, no existing formulation can be found in literature. However, this type of specification can be directly input to CPLEX as logical constraints. Therefore the correctness of the generated formulation can be verified by comparing the result with the ILOG OPL solution.

Proposition (4) can be transformed into CNF as

$$\begin{cases} \sum_{p_k \in \mathcal{P}} y_{p_k} \leq 1 \\ \sum_{p_k \in \mathcal{P}} z_{p_k} \leq 2 \end{cases} \quad \text{with } \forall p_k \in \mathcal{P}$$

$$\begin{cases} (\neg y_{p_k}, \alpha(p_k))(\neg y_{p_k}, \beta(p_k))(y_{p_k}, \neg \alpha(p_k), \neg \beta(p_k)) \\ (y_{p_k}, \neg \alpha(p_k))(y_{p_k}, \neg \beta(p_k))(\neg y_{p_k}, \alpha(p_k), \beta(p_k)) \end{cases}$$

where $\begin{cases} \alpha(p_k) \Leftrightarrow \text{CPU_overload}(p_k) \\ \beta(p_k) \Leftrightarrow \text{Mem_overload}(p_k) \end{cases}$

These CNFs can be either directly handled by ILOG OPL or transformed into generic ILP constraints based on our method. Thus, we get three ways to solve this specification with CPLEX namely F1) directly use the propositions in example 2 as constraints; F2) use the above CNFs as constraints; and F3) use the generated linear constraints by our formulation method. All of these ways are compared in the same optimization environment as used for the verification of Example 1. Eight cases with similar size but different configurations are used for the comparison. The results of the comparison is illustrated in Table 2.

Table 2: Experiment result of example 2

Idx		Size	Objective Value	No. of FSs	Time in Snd.
#1	F1	(48,7,82)	48	23	263
	F2		48	16	153
	F3		48	14	152
#2	F1	(48,7,82)	45.6	23	123
	F2		45.6	25	558
	F3		45.6	16	111
#3	F1	(48,7,82)	46.3	22	126
	F2		46.3	24	893
	F3		46.3	20	168
#4	F1	(48,7,82)	43	22	139
	F2		43	17	87
	F3		43	23	143
#5	F1	(48,8,82)	44.2	21	778
	F2		44.2	21	1713
	F3		44.2	25	566
#6	F1	(48,8,82)	42.8	20	1000
	F2		42.8	21	773
	F3		42.8	23	351
#7	F1	(48,8,82)	43.4	11	480
	F2		43.4	17	1534
	F3		43.4	17	825
#8	F1	(48,8,82)	48	22	565
	F2		48	21	802
	F3		48	35	242

In all the cases, our methods get the same result as the others. This verifies the correctness of our method. Seen from the results, out of eight cases, there are five cases where the constraints generated by our methods cost the least exploration time; five cases where our approach find the most feasible solutions; and three cases where our approach find the most feasible solutions within the least computation time. Our method introduces a reduced number of variables for the linearization, The result reflects that our method is more likely to cost less computation time or find more feasible solutions before reaching the optimal solution.

6. CONCLUSION

The main contribution of this paper is a formal and user-

intelligible method for specifying arbitrary architectural constraints for distributed embedded systems as logical formulae. The method saves designers from writing complicated mathematical equations for these constraints. Appearing as CPBPs, architectural constraints represented in our method are easy to understand and enable automatic transformation into linear inequalities acceptable by any ILP solvers. The procedure of creating the CPBP formulae and transformation to linear inequalities reduces the number of introduced variables. Thus our formal method also reduces the computation time of solving the optimization problem.

Due to the space limitation and the readability of this paper, the example used here are relatively simple and not practical enough comparing to some of the contemporary DSE methods. However our approach can be extended to a larger scale as long as the design variables are binary and the constraints can be represented as CPBPs.

This work can be improved on both application dimension and algorithm dimension. On application dimension, a user friendly domain specific language can be defined as the concrete syntax of our formal representation so as to better model architectural constraints. On algorithm dimension, 1) the generality of the formal representation can be further extended by adopting first-order logic; and 2) the efficiency of the formulation needs also to improve by considering the features of the CNF converter and the ILP solver.

References

- [1] AUTOSAR. Layered Software Architecture V4.2. Technical report, 2014.
- [2] C. Bolchini and A. Miele. Reliability-driven system-level synthesis for mixed-critical embedded systems. *IEEE Trans. on Computers*, 62(12):2489–2502, 2013.
- [3] Bosch. CAN Specification V2.0. Technical report, 1991.
- [4] R. Davis and a. Burns. Response Time Upper Bounds for Fixed Priority Real-Time Systems. *2008 Real-Time Systems Symposium*, pages 407–418, Nov. 2008.
- [5] M. Di Natale and A. Sangiovanni-Vincentelli. Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. *Proceedings of the IEEE*, 98(4):603–620, Apr. 2010.
- [6] P. Emberson and I. Bate. Stressing Search with Scenarios for Flexible Solutions to Real-Time Task Allocation Problems. *IEEE Transactions on Software Engineering*, 36(5):704–718, Sept. 2010.
- [7] IBM. ILOG CPLEX Optimizer.
- [8] S. Islam, R. Lindström, and N. Suri. Dependability driven integration of mixed criticality SW components. In *Proceedings - Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2006*, pages 485–495, 2006.
- [9] C.-W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli. Security-aware mapping for CAN-based real-time distributed automotive systems. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, volume 7, pages 115–121. IEEE, Nov. 2013.
- [10] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-p. Babau, L. Lemarchand, and S. Gerard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. *ACM SIGPLAN Notices*, 48(5):121, May 2013.
- [11] E. Niklas. Translating Pseudo-Boolean Constraints into SAT. 2:1–26, 2006.
- [12] M. Panić, S. Kehr, E. Quiñones, B. Boddecker, J. Abella, and F. J. Cazorla. RunPar: An Allocation Algorithm for Automotive Applications Exploiting Runnable Parallelism in Multicores. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis - CODES '14*, pages 1–10, New York, New York, USA, 2014. ACM Press.
- [13] B. Pfahringer. Conjunctive normal form. In C. Sammut and G. Webb, editors, *Encyclopedia of Machine Learning*, pages 209–210. Springer US, 2010.
- [14] K. W. Tindell, A. Burns, and a. J. Wellings. Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.
- [15] E. Wozniak, A. Mehiaoui, C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard. An optimization approach for the synthesis of AUTOSAR architectures. In *IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–10. IEEE, Sept. 2013.
- [16] J. Youn, I. Park, and M. Sunwoo. Heuristic Resource Allocation and Scheduling Method for Distributed Automotive Control Systems. *International Journal of Automotive Technology*, 14(4):611–624, 2013.
- [17] M. Zeller and C. Prehofer. Modeling and efficient solving of extra-functional properties for adaptation in networked embedded real-time systems. *Journal of Systems Architecture*, 59(10):1067–1082, Nov. 2013.
- [18] X. Zhang, L. Feng, D.-j. Chen, and M. Törngren. Design-Space Reduction for Architectural Optimization of Automotive Embedded Systems. In *2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)*, pages 1103–1109, New York, 2015.
- [19] Q. Zhu, Y. Yang, M. Natale, E. Scholte, and A. Sangiovanni-Vincentelli. Optimizing the Software Architecture for Extensibility in Hard Real-Time Distributed Systems. *IEEE Transactions on Industrial Informatics*, 6(4):621–636, Nov. 2010.
- [20] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems*, 11(4):1–30, Dec. 2012.