Design of Assured and Efficient Safety-critical Embedded Systems

This is the title page dummy. This page should be substituted for a real title page. The real title page will be provided by Publishing and Graphic services after having submitted your posting details in DiVA. The DiVA registration form can be found at
https://uu.diva-portal.org/dream/.

More information about the publishing routines can be found at
http://beta.ub.uu.se/pgs

Abstract page

Abstract Safety-critical systems should be analyzed rigorously to avoid undesirable consequences. Thus, the safety-critical requirements specifications should be comprehensible and consistent, and the software design should conform to the specifications. In particular, analyzing the timing specifications is not trivial especially for safety-critical software that is refined by multi-rate periodic tasks, mainly due to the undersampling/oversampling effects. When the multi-rate software is deployed on a distributed architecture, e.g., electrical/electronic vehicular system, its reliability should be maximized to counter the higher risks of failure due to the distributed computing. However, reliability engineering usually requires additional critical system resources such as power and energy. So, to accommodate current and future software functionality, the distributed safety-critical software should be efficiently deployed while meeting the timing and reliability requirements.

In this thesis, we propose formal methods and optimization techniques to assure improved quality of requirements specifications and software design, and to efficiently map software functionality to hardware. Contributions of the thesis are: (i) a constrained requirements specifications language of embedded systems, *ReSA*; (ii) a formal analysis of ReSA specifications via SMT and Ontology; (iii) a statistical model checking of a software design, modeled in Simulink, via transformation to a network of stochastic timed automata; (iv) an integrated allocation of fault-tolerant software with end-to-end timing and reliability constraints via integer integer-linear programming and hybrid particle-swarm optimization. Our proposed solutions are evaluated on automotive use cases such as the adjustable-speed limiter (ASL) and the brake-by-wire (BBW) systems from Volvo Group Trucks Technology (VGTT), and on an engine management benchmark from Bosch.

To My Parents

List of Papers Included in the Thesis

This thesis is based on the following papers:

Paper A ReSA: An ontology-based requirement specification language tailored to automotive systems. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar.*In the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)(pp. 1-10). IEEE.*

Paper B ReSA tool: Structured requirements specification and SAT-based consistency-checking. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar. *In the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS)(pp. 1737-1746). IEEE.*

Paper C Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar. *In the International Conference on Software Engineering and Formal Methods (pp. 332-348). Springer, Cham.* Acceptance rate: 26%.

Paper D SIMPPAAL - A Framework For Statistical Model Checking of Industrial Simulink Models. Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz and Henrik Lönn. *Submitted to ACM Transaction on Software Engineering and Methodology (TOSEM). ACM Journals.*

Paper E Power-aware Allocation of Fault-tolerant Multirate AUTOSAR Applications. Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Faragardi, Saad Mubeen and Cristina Seceleanu. *In the 25th Asia-Pacific Software Engineering Conference (APSEC'18). IEEE. .*

Paper F Optimized Allocation of Fault-tolerant Embedded Software with End-to-end Timing Constraints

Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Faragardi, Saad Mubeen and Cristina Seceleanu. Optimized Allocation of Fault-tolerant EmbeddedSoftware with End-to-end Timing Constraints. *Mälardalen Real-time Research Center Technical Report (MRTC).* Submitted to Elsevier JSA Journal.

List of Papers Not Included in the Thesis

- Evaluating industrial applicability of virtualization on a distributed multicore platform. Nesredin Mahmud, Kristian Sandström, and Aneta Vulgarakis. *In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), pp. 1-8. IEEE, 2014.*

- The multi-resource server for predictable execution on multi-core platforms. Rafia Inam, Nesredin Mahmud, Moris Behnam, Thomas Nolte, and Mikael Sjödin. *In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 1-12. IEEE, 2014.*

- Simulink to UPPAAL statistical model checker: Analyzing automotive industrial systems. Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. *In International Symposium on Formal Methods, pp. 748-756. Springer, Cham, 2016.*

Reprints were made with permission from the publishers.

Contents

List of Tables

List of Figures

1. Introduction

REAL-TIME systems are usually characterized by timely computations, which are bounded by *deadline*, besides correct results of the computations [10]. They are applied in many *safety-critical embedded* systems, which are specialized computer systems designed for safety-critical applications [60], e.g., the braking system inside vehicles applies proportional force on the wheel to the pressing of a brake pedal in order to slow down (or halt) the vehicle, and must act within sometime otherwise the system fails, consequently, accident can happen. Therefore, safety-critical real-time systems should be analyzed rigorously for functional and timing correctness, which is also specified in the functional safety standards, such as the ISO 26262 "Road vehicles-Functional safety" [32]. The latter standard also suggests the use of *formal methods*, which are mathematical techniques and tools that enable unambiguous specification, modeling and rigorous analysis [54], to develop safety-critical automotive systems.

In distributed computing [36], the safety-critical software is mapped on multiple hardware systems to capitalize on the computational power provided by the distributed architecture, e.g., the braking software can be executed on multiple electronic control units (ECU). Since the distributed software is normally exposed to a greater degree of permanent and transient faults, reliability of the safety-critical software should be maximized to improve dependability of the system which requires additional critical systems resource such as power and energy besides computational resources. However, the embedded hardware is usually resource constrained, therefore, the software should be efficiently mapped to the hardware to conserve critical system resources, thereby accommodate current and future growth of the software functionality.

In this thesis, we apply formal methods to improve the requirements specifications of safety-critical systems, and to analyze the functional and timing behavior of the safety-critical software against the specifications. The safety-critical specifications should be unambiguous, comprehensible, etc [1]. According to the ISO 26262 standard, semi-formal or formal languages are recommended to specify safety-critical requirements. However, natural language is the de facto method to specify embedded systems requirements in industry because it is intuitive and expressive, though inherently ambiguous [1]. In the context of natural language, template-based specification and controlled natural language can be considered semi-formal

and formal specification methods. The template-based specification methods, e.g., requirements boilerplates [31], property-specification systems [18], etc., lack meta-model to effectively create templates, and is usually cumbersome to select the templates. The controlled natural languages, e.g., Attempto [27][26], etc., renders the syntax and semantics of the natural language and have formal semantics, however lacks support for embedded systems, hence are less effective. In this thesis, we propose a constrained natural language which is domain-specific and uses the notion of boilerplates to facilitate reuse. The specifications have semantics in Boolean logic and description logic to enable rigorous analysis via Boolean satisfiability [46] and ontology [8], respectively.

The specifications are employed in subsequent system development including software design to verify the latter for correct functionality. The software design is usually modeled, simulated and analyzed before implementation. In this regard, Simulink is one of the most widely used development environment for multi-domain, multi-rate, discrete and continuous safety-critical systems in industry [33]. For this main reason, there is increasing interest in formal analysis of Simulink models [47]. Simulink Design Verifier[1], which is based on the Prover[2] model checker, is the de facto tool in the Simulink environment to formally verify Simulink design models. However, it has limited functionality, e.g., it supports only discrete models, has issues with scalability due to state-space explosion, and lacks verification of timed properties [40]. In contrast, we propose a scalable, timed analysis via a statistical model checking [39], which uses traces of executions and statistical analysis techniques, e.g., monte-carlo simulation, etc., first by transforming Simulink models into a network of stochastic timed automata using timed-automata patterns [25].

The software design should be mapped to hardware effectively, that is satisfying the timing and reliability requirements of the distributed safety-critical software, but also efficiently to minimize the power consumption of the distributed system to facilitate extensibility of the software, and also accommodates the demand from the increasing software functionality. We consider the software is scheduled using a fixed-priority preemptive policy, which is quite common in industry, and posses end-to-end timing requirements, e.g., the time duration between the brake-pedal press and the slow-down (or halt) action. Furthermore, we consider the fault tolerance as a means to maximize reliability of the distribute safety-critical software by mapping redundant software functionality on different computing units. We propose *exact* and *heuristic* optimization methods, which deliver optimal and near-optimal solutions, respectively, to efficiently map the distributed safety-critical software to a network of computing units. Specifically, we propose a formulation of integer-linear programming

[1]Simulink Design Verifier - https://se.mathworks.com/products/sldesignverifier.html
[2]Prover - https://www.prover.com/software-solutions-rail-control/formal-verification/

(ILP) [43], which is solved using branch and bound. Furthermore, we propose a hybrid-particle optimization [50], which is a meta-heuristic algorithm, to solve the shortcomings of the exact method for large-scale problems [44] with trade-off over non-optimality.

- **Formal Analysis of natural language requirements:** we propose a fairly expressive, flexible yet structured and domain-specific constrained natural language, called *ReSA* [41][45]. The language has semantics in Boolean and description logic to support for shallow and rigorous analysis, respectively. The Boolean specifications are checked for consistency using the satisfiability-modulo theory via the Z3 SMT solver. Whereas, the description logic is used to encode the specification as ontology, where we check consistency of the specifications at the lexical level using Reasoner (Inference engine) such Hermit. The ReSA tool, which consists of an editor and implements consistency-checking functionality, is integrated seamlessly into EATOP, which is an open source EAST-ADL IDE, to complement the requirements modeling.

- **Scalable analysis of Simulink models:** we propose a pattern-based, execution-order preserving automatic transformation of atomic and composite Simulink blocks into stochastic timed automata that can be formally analyzed using UPPAAL Statistical Model Checker [9]. Our method is scalable, and has been validated on industrial use cases [23]. The statistical model checker analyzes a state-transition system by conducting statistical analysis on the collected traces of the system executions, effectively mitigating the state-space explosion of (exact) model checking [39].

- **Efficient Power consumption ILP and metaheuristics:** we propose an integer-linear programming (ILP) model to the allocation of distributed software on the network of heterogeneous computing units, which have different processor speed, failure rate and power consumption specifications. The ILP implemented in JAVA using the ILOG CPLEX interface, and subsequently solved the CPLEX solver.

- **Validation on industrial use cases:** Our contributions such as its the ReSA language as well as the proposed formal analysis of Simulink model is validated on industrial use cases, which are provided

Our solutions are evaluated on industrial automotive use cases and on a realistic benchmark. The formal analysis of the natural language requirements specifications in ReSA and the formal analysis of Simulink models are evaluated on the adjustable speed-limiter (ASL) and brake-by-wire (BBW) systems provided by Volvo Group Trucks Technology (VGTT). ASL is a

speed-limitation automotive function which controls the vehicle speed of Volvo trucks from speeding up, and is useful in roads where speed-limitation signs are in place. The ASL use case consists of around 300 requirements, which are specified in natural language, architectural models in EAST-ADL and Simulink models. The integrated software allocation is evaluated on the engine management system benchmark provided by Bosch [] provided for AUTOSAR applications. The benchmark consists of statistics of the schedulable objects, such as mean values, shares of timing specifications and activation mechanisms of the schedulable objects in the system.

1.1 Thesis Outline Overview

The thesis is divided into two parts. The first part is a summary of our research. It is organized as follows: in Chapter 2, we give the background information on description logic, Boolean satisfiability problem, Simulink, stochastic timed automata, and meta-heuristic optimization. In Chapter 3, we explain the research problem and outline the research goals. The thesis contributions are discussed in Chapter 4, followed by the related work in Chapter 5. In Chapter 3, we describe the research method applied to conduct the research. Finally, in Chapter 7, we conclude the thesis and outline possible directions for future work.

2. Background

In this section, we explain the background about the concepts and technologies imployed in the thesis. These includes the logic-based reasoning methods such as the Boolean satisfiability (SAT) and ontology-based on the description logic which are employed to improve quality of requirments specifications, the UPPAAL statistical model checking and stochastic time automata which is imployed in the formal analysis of Simulink models, and the integer-linear programming and metaheuristics optimization methods which are imployed in the resource optimization in the software to hardware mapping.

2.1 Logic-based Reasoning

In this thesis, we express requirements specifications in Boolean logic and apply Boolean satisfiability to detect inconsistencies in the specifications. Moreover, we apply ontology for a more rigoruous analysis of thes specifications.

Boolean Satisfiability (SAT)

The study of a Boolean formula is generally concerned with the set of truth assignments (assignments of 0 or 1 to each of the variables) that make the formula true. Finding such assignments by answering the simpler question: "does there exist a truth assignment that satisfies the boolean formula?" is called the boolean satisfiability problem (SAT), which is proven to be NP complete. However, heuristics and approximation algorithms do exist to return the result. A SAT problem is formally defined as follows [17].

Z3 Theorem Prover/ SMT Solver. Satisfiability Modulo Theories (SMT) [7] is a decision problem expressed in the first-order logic which integrates a collection of background theories, e.g., on real numbers, bit vector, array, etc. SAT problems are usually described in terms of satisfiability, which is checked using decision procedures, known as SAT solvers. Z3 is a well-known SMT solver as well as a theorem prover developed by Microsoft Research which integrates several background theories and decision procedures using a DPLL- based SAT solver [15]. It has been used in the formal verification and reasoning of software and hardware systems. Further, it has a strong support

for integra- tion to verification tools using its well-known APIs, written in Java, Phyton, C++, C and .Net.

The Z3's input language is an extension of the SMT-LIB2 standard [**?**] which is used to assert formulas (expressions) as commands. The formulas can be checked for satisfiability using the command (check-sat). If the formulas are satisfiable, Z3 returns sat; if the formulas are not satisfiable, Z3 returns unsat, or if it cannot decide it returns unknown. In case of unsat, if the unsat-core option is enabled, Z3 returns the unsatisfiable subsets of assertions using the (get-unsat-core) command.

Ontology based on Description Logic

Ontology is a knowledge-base representation method frequently used in the artificial intelligence and other areas to facitate decision making. Description Logic (DL) [2] has its origins in the 1970s, and is intended to address the need for logical support to the representation of knowledge bases, mostly designed by using frames and semantic networks. Currently, the language is predominantly used in web semantics (e.g., OWL) [**?**, **?**, **?**], artificial intelligence [6], bioinformatics [55], software engineering, natural language pro- cessing. The language is designed with practicality in mind; consequently, it is usually backed by solid reasoning services that terminate and deliver results in reasonable time. The language uses Concepts (unary predicates), Roles (binary predicates) and Instances (logical symbols) that are recursively constructed us- ing Constructors. The most common constructor Subconcept (or concept inclu- sion) $\sqsubseteq$, is used to define concept hierarchies, and the equivalence constructor $\equiv$ defines the equivalence of concepts, through the bidirectional concept inclu- sions. Other basic constructors of the language include the conjunction (intersection) $\sqcap$, disjunction (union) $\sqcup$, the existential quantifier $\exists$, and the universal quantifier $\forall$. Currently, there are many variants of the language that differ in their expressivity. (Attribute Language Complement) is considered the basis of the many Description Logic variants, for instance, $\mathcal{SROIQ}$ which is implemented in OWL 2 extends $\mathcal{ALC}$ (abbr. $\mathcal{S}$) with the role inclusion ($\mathcal{R}$), nominal ($\mathcal{O}$), inverse role ($\mathcal{I}$), qualified cardinality restrictions ($\mathcal{Q}$).

2.2 Simulink Analysis

Simulink [33] is a graphical development environment for the modeling, simulation and analysis of embedded systems which is widely used in industry to model and inspect the dynamics of systems before implementation. It is robust as it supports multi-domain, continuous, discrete, hybrid systems, also discrete systems that execute with different sampling times (or multi-rate). Figure 2.1 shows a multi-rate subsystem of the

brake-by-wire Simulink model that models the brake pedal (i.e., continuous behaviror) and global brake functionality (i.e., discrete behavior), which executes every 10ms and 20ms.

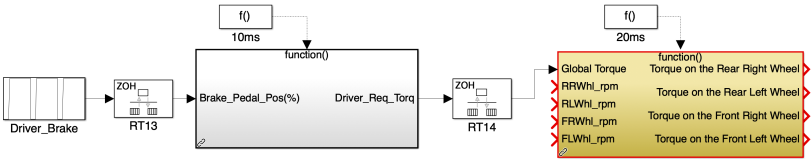


Figure 2.1: Brake pedal and global brake controller of the brake-by-wire model.

Simulink is frequently used in the development of safey-critical systems, e.g., automotive, avionics, furthermore, it is used to generate code automatically from the models. Therefore, it is crucial that such models are analyzed rigorously. The Simuink Design Verifier (SDV) provides a formal verification technique that is based on the exact model checking to exhaustively verify functional properties [48], however, its funtionality is limited as it lacks support for timed anlaysis, which is vital to ensure predictability of safety-critical embedded systems.

Simulink Blocks

A Simulink model is constructed from communicating function blocks (or Simulink blocks). The blocks implement simple to complex functionlity and consist of Input and Output ports, which enable communication via connectors. The latter modeling elements support basic and user-defined datatypes, e.g., integer, floating-point; and simple and complex data structures, e.g., scalar, vector, matrix. Simulink blocks are classified into *virtual* (non-computational, e.g., Mux, Demux blocks) and *non-virtual* (computational, e.g., Gain, Integrator) blocks based on their computability. The non-virtual blocks improve the visualization of the model, but unlike the non-virtual blocks, they are not executed, hence do not affect the execution semantics of the model. The blocks can be composed into groups, e.g., using Subsystem, Model blocks, to enable hierarchical modeling, which is used to improve the visualization, and to enforce execution order on a group of blocks. The fundamental constructs of the composite blocks are *atomic* blocks, e.g., Gain, Sum blocks.

The non-virtual (computational) atomic blocks can be categorized into *continuous* and *discrete* blocks based on the execution semantics of the blocks. A discrete block executes periodically with sample time t_s, whereas a continuous block executes over infinitesimal sample times. Since the Simulink blocks libraries are not usually sufficient to model practical embedded system systems, Simulink supports mechanisms to extend functionality that engineers can exploit to develop complex systems. The

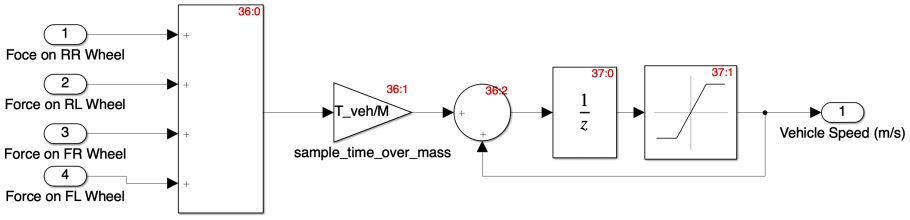


Figure 2.2: A Subsystem block from the brake-by-wire model, computes vehicle speed. The labes in red color are indexed identifiers and denote of the execution order.

mechanisms include S-function, Custom Block and Masking. S-function is a computer language of Simulink blocks which allows advanced implementations of block routines, written in MATLAB, C, C++, or Fortran.

Execution of Simulink Blocks

Duirng the initial phase of the simulation, the model is compiled, thus the order in which the blocks are executed is established known as *sorted order* list. Figure 2.2 shows the execution order via the labels in red color, annotated as $s : b$, where s denotes the system/subsystem index and b the block index[1]. Basically, the list is determined according to the data dependency of blocks' outputs on the blocks' input ports, i.e., if the output depends on the current value of the input, the input port is identified as *direct-feedthrough* port. Thus, to preserver the data dependency in the model, the sort order rules require that the blocks that derive other blocks that have direct-feedthrough ports must come first in the list, e.g., blocks that derive Gain block. However, the blocks with non-direct-feedthrough ports, e.g., Delay block, can execute in any order, considering the previous rule. The execution order is also affected by the user defined priorities, neverthless, the priorites do not violate the rules. The sorted order list can be fetched by simulating the model in the debugging mode.

2.3 UPPAAL Statistical Model Checking

UPPAAL SMC [**?**] is a statistical model checker (implemented as an extension of UPPAAL toolset) for system models represented as networks of stochastic priced timed automata. In UPPAAL SMC the automata have a stochastic interpretation based on: (i) the probabilistic choices between multiple enabled transitions, and (ii) the non-deterministic time delays that can be refined based on probability distributions, either uniform distributions for time-bounded delays or user-defined exponential distributions for unbounded delays.

[1] https://se.mathworks.com/help/simulink/ug/controlling-and-displaying-the-sorted-order.html

Stochastic Timed Automata

A *stochastic priced timed automaton* (SPTA) is defined as the following tuple:

$$SPTA = \langle L, l_0, X, \Sigma, E, R, I, \mu, \gamma \rangle, \tag{2.1}$$

where L is a finite set of locations, $l_0 \in L$ is the initial location, X is a finite set of continuous variables, $\Sigma = \Sigma_i \uplus \Sigma_o$ is a finite set of actions partitioned into inputs (Σ_i) and outputs (Σ_0), E is a finite set of edges of the form (l, g, a, φ, l'), where l and l' are locations, g is a predicate on $\mathbb{R}^X$, action label $a \in \Sigma$, and φ is a binary relation on $\mathbb{R}^X$, $R : L \to \mathbb{N}^X$ assigns a rate vector to each location, I assigns an invariant predicate $I(l)$ to any location l, μ is the set of all density delay functions $\mu_s \in L \times R^X$, which can be either uniform or exponential distribution, and γ is the set of all output probability functions γ_s over the Σ_o output edges of the automaton.

The semantics of the probabilistic SPTA is defined over a timed transition system, whose states are pairs $s = (l, v) \in L \times \mathbb{R}^X$, with $v \models I(l)$, and transitions defined as: (i) delay transitions ($(l, v) \xrightarrow{d} (l, v')$ with $d \in \mathbb{R}_{\geq 0}$ and $v' = v + d$), and (ii) discrete transitions ($(l, v) \xrightarrow{a} (l', v')$ if there is an edge (l, g, a, Y, l') such that $v \models g$ and $v' = v[Y]$, where $Y \subseteq X$, and $v[Y]$ is the valuation assigning 0 when $x \in Y$ and $v(x)$ otherwise). We write $(l, v) \rightsquigarrow (l', v')$, if there is a finite sequence of delay and discrete transitions from (l, v) to (l', v'). The delay density function μ_s over delays in $\mathbb{R}_{\geq 0}$ for each state s is either a uniform or an exponential distribution, depending on the invariant of location l. Let E_l denote the disjunction of guards such that $(l, g, o, -, -) \in E$ for some output o. Then, $D(l, v) = \sup\{d \in \mathbb{R}_{\geq 0} : v + d \models I(l)\}$ denotes the supremum delay, whereas $d(l, v) = \inf\{d \in \mathbb{R}_{\geq 0} : v + d \models E_l\}$ denotes the infimum delay before enabling an output. If $D(l, v) < \infty$ then the delay density function μ_s for a given state s is a uniform distribution over the interval $[d(l, v), D(l, v)]$, otherwise it is an exponential distribution with a rate $P(l)$. For every state s, the output probability function γ_s over Σ_o is a uniform distribution over the set $\{o : (l, g, o, -, -) \in E \wedge v \models g\}$ whenever the set is non empty.

Under the assumption of input-enabledness, disjointedness of clock sets and output actions, a collection of composable SPTA can be defined as a *network of SPTA* (NSPTA), via the parallel composition operator ($A1 \parallel A2 \parallel ... \parallel An$). The states of the NSPTA are defined as a tuple $s = \langle s_1, ..., s_n \rangle$, where s_j is a state of A_j of the form (l, v), where $l \in L^j$ and $v \in \mathbb{R}^{X^j}$, where different automata synchronize based on standard broadcast channels. The probabilistic semantics is based on the principle of independence between components. Each component decides on its own (based on a given delay density function and the output probability function) how much to delay before producing an output.

For encoding the patterns presented in this paper, we use SPTA with real-valued clocks that evolve with implicit rate 1. These automata are in fact timed automata with stochastic semantics, called *stochastic timed automata* (STA). A *network of STA* (NSTA) is a parallel composition of STA, defined in a similar way as NSPTA. The notion of SPTA is introduced due to the fact that, for analysis we use monitor automata (composed in parallel with the actual system model) that implement the *stop-watch* mechanism, which renders the model a network of stop-watch timed automata, which is a subset of NSPTA.

Properties Specification

UPPAAL SMC uses a probabilistic extension of *weighted metric temporal logic* (WMTL) [**?**] to provide:

- *Hypothesis testing*: check if the probability to reach a state ϕ within cost $x \leq C$ is greater or equal to a certain threshold p ($Pr[<= bound](\star_{x \leq C} \phi) \geq p$),
- *Probability evaluation*: calculate the probability $Pr[<= bound](\star_{x \leq C} \phi)$ for some NSPTA,
- *Probability comparison*: is $Pr[<= bound](\star_{x \leq C} \phi_1) > Pr[<= bound](\star_{y \leq D} \phi_2)$?,

where $\star$ stands for either *future* ($\Diamond$) or *globally* ($\Box$) temporal operator, and *[<= bound]* denotes the time bound of the executions.

2.4 Integer-linear Programming (ILP)

ILP is an optimization problem where the decision variables are integer, and the objective function and the constraints are linear models. They are problems that can be represented as

$$Maximize \sum_{j=1}^{n} c_j x_j \tag{2.2}$$

Subjec to: (2.3)

$$\sum_{j=1}^{n} a_i j x_j \leq b_i \qquad \text{for all } i = 1, ..., m \tag{2.4}$$

$$x_j \geq 0 \wedge x_i \in \mathbb{I} \qquad \text{for all } i = 1, ..., m \tag{2.5}$$

It has many mathematical, and engineering applications, e.g., transportation, scheduling. The binary (0-1) problem is a special case of ILP where the decions variables takes on 0 or 1. It appied in several decision making problems, e.g., resource allocation in the scheduling/ assignment

problem. The software mapping is an instance the assignment problem where each binary variable denote if a software component is mapped to a processor (or computing unit) or not, i.e., 1 if assigned otherwise 0.

ILP problems are frequenlty solved via exact algorithms, e.g., branch and bound, but can also be delt via heuristics, e.g., using simulated annealing, hill climbing. In this work, we use the CPLEX solver form IBM to the software-to-hardware mapping optimization.

2.5 Population-based Metaheuristics

For complex optimization problems, the ILP formulation is a difficult task and the scalability to solve large problems instances is usually prohobitively expensive. Metaheuristics is a type of heuristics which uses search strategies that are sometimes less dependent on the problems and more efficient computation-wise but less optimal. Population-based metaheuristics is a class of meta-heuristic methods which uses a set of individuals (or population) in the search strategy to determine the global optima of the problem, e.g., genetic algorith, differential evolution, particle swarm optimization.

In this work, we apply differential evolution, particle-swarm optimization to search for the best software-to-hardware mapping solution. The differential evolution employes a set of agents, which represent candidate solutions, to find the best software-to-hardware mapping solution. Each agent applies a particular types of evolutionary operators such as mutation, crossover and selection to reach a target position in the search space. A mutant agent (or offspring) is generated for every agent and in every iteration from three other angents. If the offspring is more fit, i.e., better than the parent agent, it replaces the latter, otherwise is discarded.

$$\mathbf{v} \leftarrow \mathbf{a} + F \circ (\mathbf{b} - \mathbf{c}) \tag{2.6}$$

$$\mathbf{u} \leftarrow crossOver(\mathbf{v}, \mathbf{x}, CF, F) \tag{2.7}$$

$$\mathbf{x} \leftarrow \begin{cases} \mathbf{u} & \text{if } f(\mathbf{u}) < f(\mathbf{x}) \text{ functions} \\ \mathbf{x} & \text{otherwise} \end{cases} \tag{2.8}$$

where $F \in [0,2]$ is the differential weight, $CF \in [0,1]$ is the crossover probability.

Likewise, in the particle-swarm optimization, the method employs agents (in this case, particles), which are memeber of the population and represents candidate solutions. It records the best position of each particle so far $\mathbf{p}_{bst}$ and best position of the population $\mathbf{z}$. Thus, the motion of each particle is guided by its velocity, attraction towards its best position $\mathbf{p}_{bst} - \mathbf{p}$ and attraction towards the best position of the swarm $\mathbf{z} - \mathbf{p}$, where $\mathbf{p}$ is a n-dimensional matrix which

represents the current position of a particle in the search space.

$$\mathbf{v} \leftarrow \omega\mathbf{v} + c_1 Rand() \circ (\mathbf{p}_{bst} - \mathbf{p}) + c_2 Rand() \circ (\mathbf{z} - \mathbf{p}) \tag{2.9}$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v}, \tag{2.10}$$

where ω is the weight of the velocity, also known as *inertia coefficient* and controls the convergence of the algorithm. The c_1, c_2 constants are acceleration coefficients and control the weight of attraction towards the cognitive and social components, respectively. $Rand() \in U(0,1)$ is a random function along the acceleration coefficients, which is element-wise multiplied with the components to improve diversity of the search by introducing stochastic behavior.

3. Problem Formulation

The automotive electrical/electronic system executes complex safety-critical software, e.g., x-by-wire software, engine control, traction control, etc. Over the last decades, the complexity of the safety-critical software has been on the rise which is evident on the modern cars, which implement many and complex automotive functions, and also on the emergence of the electrical and autonomous vehicles. Thus, the thesis is motivated by the need for advanced (or rigorous) methods to the requirements specification, modeling and analysis of the complex safety-critical automotive software, and their seamless integration into the existing methods and tools of the automotive systems development at VGTT and Scania. Furthermore, the thesis is motivated by the need for efficient mapping of safety-critical software to hardware in the distributed computing to facilitate software extensibiliy and support the increasing functionality of the automotive systems.

Thus, the *overall goal* of the thesis is to:

> **Overall Goal** — Provide assurance of safety-critical functionality, at the various levels of abstraction, via formal analysis, and optimization of critical system resources.

The overall goal is refined via *research goals*, which state the needs or concerns that the thesis should address and are formulated as follows:

3.1 Research Goals

Many safety-critical automotive systems are developed according to the ISO 26262 standard, which recommends highly the use of semi-formal languages to specify safety-critical requirements to improve quality of the specifications, e.g., by reducing ambiguity and improving comprehensibility. In the context of textual representations, the semi-formal specification methods are constrained natural languages, such as templates (e.g., requirements boilerplates [20][45]), controlled natural languages [38](e.g., Attempto [27]).

The template-based methods inherently lack meta-model (or grammar), therefore is difficult to add new templates effectively, moreover, template selection is usually cumbersome. The existing controlled natural languages lack effective support of specifying embedded systems requirements.

Thus, the first research goal is to:

> **RG 1:** — Reduce ambiguity and improve the comprehensibility of natural-language requirements using domain-specific knowledge of embedded systems.

One of the mechanisms to improve natural language specifications is by constraining the language, including its syntax, semantics and the lexicon [38]. The design of a constrained natural language for the specification of requirements is not trivial. By constraining the language, its expressiveness and intuitiveness can be impaired [1][52], therefore, appropriate trade-offs should be made during the design in order to have a robust and effective specification language.

Besides improving quality of individual requirements, the latter should be analyzed in ensemble in order to detect errors that span multiple specifications, e.g., logical contradictions. However, natural language lacks formal (or precise and unambiguous) semantics, therefore is difficult to rigorously analyze (or reason) natural-language requirements specifications. There are several methods to natural language semantics, of which the use of *logic* is common [12].

Thus, the second research goal is to:

> **RG 2:** — Facilitate formal analysis of the requirements specifications through transformation to Boolean and description logics.

Natural language specifications are constructed from syntactic units, such as words, phrases, clauses, statements, etc. Consequently, rigorous analysis of the specifications involve parsing and interpreting the syntactic units, which is a complex problem in computational linguistics [12]. The depth of the interpretation (or semantics) greatly affects the applicability of the methods, e.g., the propositional logic representation of the specifications is simple and the analysis scales well, however, it is shallow as it abstracts away the details. On the other hand, the first-order-logic representations are more rigor, thus enable thorough analysis but are less tractable. Therefore, appropriate interpretation of the natural language specifications is crucial.

The software designs and software-design units (or behavioral models) should conform to the requirements specifications. We consider the software-design units are modeled in Simulink, which is the most widely used model-based development environment in industry to model and simulate the behavior of multi-domain, discrete, continuous embedded systems. Simulink also enables the generation of code from discrete Simulink models which directly execute on specific platforms, thus is crucial to

conduct rigorous analysis of the Simulink models to reduce errors introduced in the generated code.

The de facto Simulink analysis techniques, e.g., by type checking, simulation, and formal verification via the Simulink Design Verifier (SDV[1]) are not sufficient to address the full correctness of safety-critical real-time Simulink models. SDV lacks support for checking temporal correctness as specified in timed properties, e.g., in TCTL, and also lacks support for verifying continuous models and suffers from scalability due to its reliance on the exact model-checking [40]. In contrast to the exact model checking, the statistical model-checking verifies properties over sufficiently collected traces of system simulations via statistical methods. It scales better over the trade-off for exhaustiveness.

Thus, the third research goal of the thesis is to:

> **RG 3:** — Enable formal analysis of large-scale, multi-rate and hybrid Simulink models using statistical model-checking.

Simulink consists of connected and hierarchical Simulink blocks, which encode mathematical functions [33]. For industrial systems, the number of blocks in a Simulink model can be in the order of thousands, and the blocks can be triggered with different sampling frequencies for discrete blocks and without any sampling frequency for continuous blocks. Therefore, typical industrial Simulink models are usually complex and comprise mixed signals, multiple rates, discrete and continues Simulink blocks, making the model checking challenging.

In the distributed computing, the automotive software is allocated on multiple computing units (or ECU), consequently is exposed to higher permanent and transient faults, hence necessitates to maximize the reliability of the safety-critical software system. Fault tolerance using redundancy is the most widely approach to improve reliability such as by replicating software functionality on multiple ECU, however, it requires additional critical system resources such as power sources. In this regard, the software-to-hardware allocation plays a crucial role to minimize the power consumption of fault-tolerant distributed safety-critical software while satisfying the timing and reliability constraints of the safety-critical software.

Thus, the fourth research goal is to

> **RG 4:** — Minimize the power consumption of distributed safety-critical software while satisfying the timing and reliability constraints during the software allocation.

The software allocation model is not trivial as we consider exact method of schedulability analysis and reliability calculations, which makes the

[1] https://se.mathworks.com/products/sldesignverifier.html

optimization complex. We assume a sufficient and necessary scheduling test based on the worst-case response time analysis [4][14], moreover end-to-end delay analysis based on the age delay semantics [51], which is practical but computationally expensive. For the reliability analysis, we apply an exact method of calculation based on the state enumeration, in contrast to the series-parallel method, which is trivial and computationally less expensive. As a result, we consider an exact optimization method using ILP for relatively small and metaheuristics for relatively large software allocation problems.

In order to show the validity of our proposed solutions, a working prototype should be developed and should also be evaluated on industrial uses cases. The validation should consider scalability and engineer-friendliness of methods and tools besides effectiveness.

Thus, the last research goal is to:

> **RG 5:** — Provide automated and engineering-friendly support for the requirements specification, software allocation of embedded and formal analysis of Simulink models.

Seamless integration of our proposed methods and tools into the existing development process require close cooperation between the domain experts and the practitioners. The role of the domain experts should be to simplify usage of the tools, e.g., by rendering their interface to existing once, etc., and the practitioners should cooperate with materials that assist the validation of the proposed solutions. The cooperation is not trivial considering the challenge of formal methods, and companies culture for being restrictive.

4. Thesis Contributions

The general contribution of the thesis is a safety-critical design approach that employs formal methods at various stages of software development such as requirements specification and software design, and a power-efficient mapping of software to hardware while satisfying timing and reliability of the software in a distributed environment. It satisfies the research goals explained in Subsection 3.1 via the following contributions: a requirements specification language tailored to embedded systems [45][41], formal analysis of the specifications [41][42] via formal and knowledge-based methods, formal analysis of large-scale Simulink models [25] and efficient mapping of software to hardware in the context of distributed architecture [43][44] via integer-linear programming and metaheuristics.

Figure 4.1 illustrates the workflow of an embedded software development that make of use our contributions. The workflow is contextualized in the automotive software development where Simulink, EAST-ADL and AUTOSAR architectural language are used.

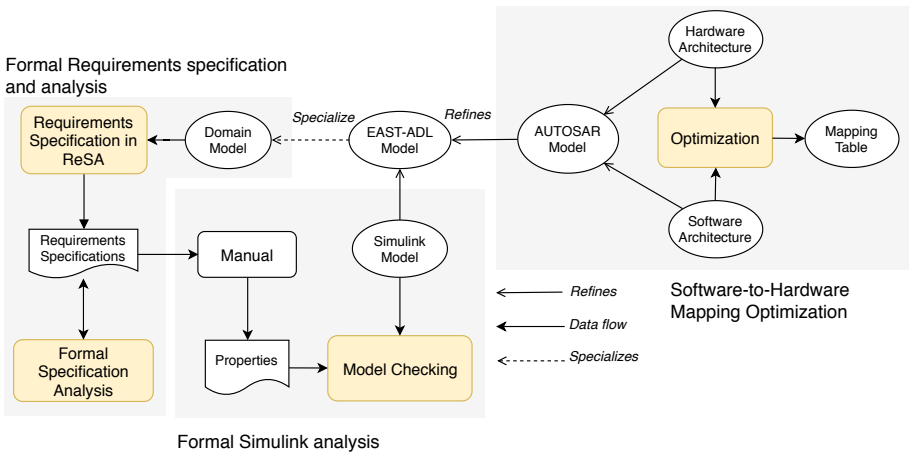


Figure 4.1: Thesis contributions workflow.

Initially the textual requirements are specified in our domain-specific language ReSA, which is a constrained natural language that is specific to embedded systems. The ReSA editor supports content completion as it is connected to a system model, which is generic. However, If the ReSA editor is required to be integrated into domain-specific architetural models, the

latter specializes the ReSA concepts, hence enabling domain-specific access. To check the consistency, the ReSA specifications are translated into formal specifications such as Boolean and description logic, subsequently, analyzed via SAT solver and inference engine (or reasoner), respectively.

After the quality of the specifications are improved, i.e., become unambiguous, comprehesible, consistent, they can be used in the verification of behaviral software designs, thus reducing later errors detections and maintainance. In this work, we consider the behavior models that are designed in Simulink, which is one of the most widely used environment for modeling, simulation and anlaysis of embedded systems. In order to rigoruously analyze a Simulink model, it is transformed into a formal model, subsequently, the latter is analyzed via statistical model checking against properties initially expressed in ReSA, and manually translated into the query (properties) language of the model checker.

The Simulink model, at the architectural level, is represented by a system architecture, which consists of software and hardware architectures. Note that, the relation between the Simulink model and the system architecture is by refinement, and is mostly manual, which is also the case in this work. The software architecture complements the behaviora model with computational resource specifications, e.g., CPU, memory, and other system critical resouce specifications, e.g., power consumption. In this regard, the software to hardware mapping can be efficiently made via optimization techniques. Our proposed optimization techniques incorporate the timing requirements of the software. Although the input model to the optimization is an AUTOSAR model, the approach can be applied to any component-based embedded software devevelopment through minor modification.

In the next subsections, we discuss the contributions in detail.

4.1 ReSA - a Constrained Requirements Specification Language of Embedded Systems

The language is designed to improve comprehensibility and reduce ambiguity of embedded systems specifications. It is a constrained natural language on both syntax and semantic levels. It employes embedded system concepts such as System, Parameter, Device, State, Mode, Entity, etc, to identify words/phrases of specification and realtions between between instance of the concepts to enforce semantics (or interepretations) at the syntactic level, e.g., the fact is a system can be activated, but not parameters, a user is notified, but not a system.

The syntactic rules allow construction of valid statements, reduce ambiguity and improve readability. The valid instantiation of the ReSA langage are sentencial forms (or requirements boilerplatrs), as shown in Table 4.1. The specification is finalized by filling the vairable elements of

the boilerpaltes, which the syntax Word|"Words": *Type*, e.g., "ASL speed control": *parameter*, ASL: *system*, as illustrated by examples in Tables 4.2. In the ReSA editor, the boilerplates can be created for reuse.

4.1.1 Syntax

Boilerplate	Description
Simple	instantiates a simple statement, e,g., `Simple := System Verb within Quantity TimeUnit.`
Compound	instantiates a compound statement, e.g., `Compound := Simple and/or Simple+`
Complex	instantiates complex statement, and is constructed from Simple and a subordinating conjunction, such as when, while, until, e.g., `Complex := if Simple, Simple`
Nested-Complex	instantiates nested-complex statements, and is constructed from independent clause and complex boilerplate(s), e.g., `Ncomplex := after Simple, Complex.`
Conditional	instantiates conditional statements including if, if-else, if-else-if

Table 4.1: *Requirements boilerplates syntax.*

R1	ASL:system shall send "the driver"*:user* notification:status every 200ms.
R2	if driver selects “ASL speed control”*:mode* and (vehicle is in “pre-running” mode or vehicle is in “running” mode) then ASL:system shall be enabled within 200ms and “ASL enabled”*:status* shall be presented to “the driver”*:user* endif
R3	After ASL:system is enabled, if IncButton*:inDevice* is pressed, ASL:system shall be activated.
R4	if driver selects “ASL speed control”*:mode* then ASL*:system* shall be disabled.

Table 4.2: *Examples of requirments specified in ReSA.*

4.2 Formal Analysis of ReSA Specifications

The ReSA specifications are transformed into Boolean and propositional logics to conduct rogorous analysis besides syntax and type checking such as consistency checking. The Boolean expressions are check for consistency via a SAT solver, and the description-logic specifications via an ontology enference engine.

4.2.1 SAT-based Analysis

The specifications are translated into Boolean expressions, subsequently, an SMT solver is employed to check the consistency of the specifications. The analysis is scalable as the SAT solving is scalable to thousands of propositional variables, though shallow since the details of each clause in the specifications are abstracted away by a propositional variable. Thus, advanced analysis is not possible or suitable using the SAT approach.

Definition 4.2.1 (Inconsistency of specifications)**.** Let $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ is the set of requirements specifications after translating ReSA specifications into Boolean expressions, where $\Phi \in N$ is a Boolean formula and denotes a requirement specification. Thus, the set Boolean expressions Φ is inconsistent if the expression $\Phi_1 \wedge \Phi_2 \wedge \cdots \wedge \Phi_n \Rightarrow false$ holds, that is, there exists at least one expression that cannot be satisfied, i.e., $\Phi_i \models false$.

Example 1. Figure 4.2 illustrates the translation of the ReSA specifications R1, R2, R3, R4, into Boolean expressions, encoded as Z3 (or SMT-LIB) format. The expressions are fed into the Z3 solver, which returns the unsat-core feedback "unsat (R1, R2 R1_Assertion)". The unsat-core shows that R2, R4 and the R2 assertion are the source of the inconsistency. Note that, Z3 only tracks labeled assertions.

The SAT-based analysis, although, easy and scalable, does not enable rigoruous analysis at the lexical level, due to lack of adequate semantic relations defined between words/phrases. This is due to the fac that, the propositional variables abstract away these details. In the next subsection, we employ ontology-based specification to capture the syntax, and interpretation of the specifications based on the *event-based* linguistic approach [42].

4.2.2 Ontology-based Analysis

The event-based semantics, which is based on the first-order logic, uses an existentially quantified events to relate words/phrases (or arguments of predicate) , clauses, adjuncts of a sentence [42], e.g., the clause “ASL:system shall limit “vehicle speed”:parameter” is represented as $\exists e$[limiting(e) & Agent(ASL) & Recipient(vehicle speed)], where Agent and Recipent are known as *thematic* roles, which define the semantic roles of the arguments

```
(declare-const p10 Bool) ; driver selects “ASL speed control”:mode
(declare-const p11 Bool) ; ASL:system shall be disabled

; Equivalence assertions
(assert (= p5 p7 (not p11)))
(assert (= p2 p10))

; Requirements assertion
(assert (! (= p1 true) :named R1))
(assert (! (=> (and p2 (or p3 p4)) (and p5 p6)) :named R2))
(assert (! (=>(and p7 p8) p9) :named R3))
(assert (! (=> p10 p11) :named R4))

; Assert conditions are true, that is, one at a time
; If the specifications are satisfiable for all the assertions
; then specifiaction is consistent
(assert (! (= (and p2 (or p3 p4)) true) :named R1_Assertion))
(check-sat)
;(get-model)
(get-unsat-core)


unsat
(R2 R4 R1_Assertion)
```

Figure 4.2: The ReSA specifiactions R1- R4 encoded as Z3 format, and the unsat-core feedback from the Z3 solver, to localize source of the inconsistency.

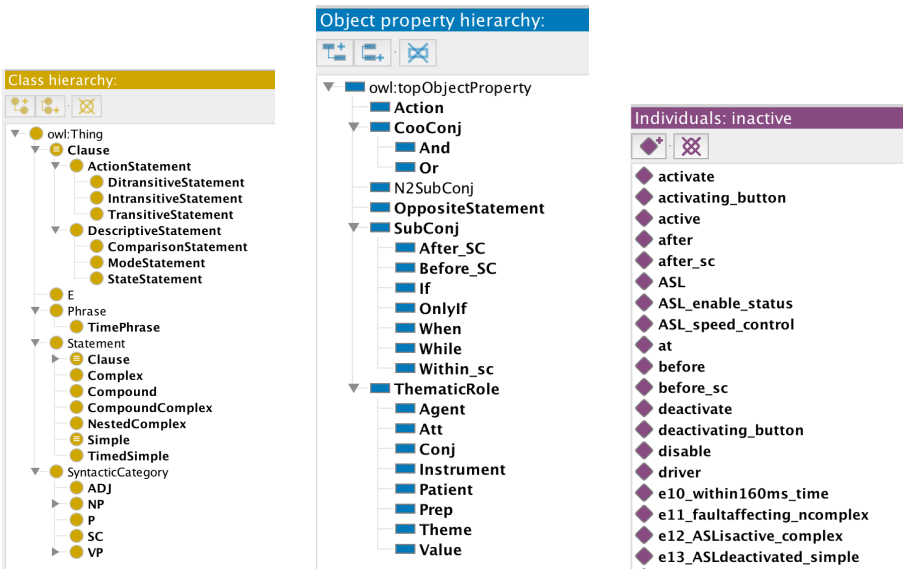
in the clause. Some of the thematic roles applied in this work are shown in Table 4.3. Since the requrement specifications entail techical words/phrase, the thematic roles of these arguments are not obvious, so we define relations between the embedded system concepts shown in Tables **??** to the thematic roles define in Table 4.3, e.g., any instance of System or User is Agent, similarly instances of Parameter is Recipient, etc. In this way, the interepreatation of the clauses become domain-specific.

Thematic Role	Description
Agent	initiates and carries out an action, and exists independently of the action.
Attribute	is the property of an entity.
Patient	undergoes state changes and exists independently of the action.
Theme	similar to Patient, but doesn’t undergo state changes.
Recipient	is the destination of an action.
Instrument	is used by an agent to cause an action, and exists independently of the action.

Table 4.3: *Thematic roles.*

A complex requriements specification can be constructed via the composition of events, e.g., “after ASL button is pressed, ASL shall limit “vehicle speed” is represented as $\exists e_1$[pressing($e1$)&Recipient(ASL button)& $\exists e_2$[activating(e_2)&Agent(ASL)&Recipient(vehicle speed)]& after(e_1, e_2)].

Followoing the event-based semantics, the ReSA specifications are translated into description logic as ontology, which is a knwoledge representation approach frequently used in artificial intelligence, and information system, e.g., the world-wide web. The ontology contains terminological assertions and concrete assertions, respectively assert, the logical structure of sentences according to the event-based semantics, and the requrements specifiactions, such as the types, thematic roles of words/phrases, respectively. Figure 4.3 the concepts, relations and instances of the requirments specifiaction ontology.



(a) Concepts. *(b)* Object properties. *(c)* Individuals.

Figure 4.3: Requirements specification ontology (screenshot from Protégé tool).

In order to strengthen the analysis, lexical relations between arguments are introduced such as anonyms, synonyms, generalization, specialization, e.g., the fact that "enabled" is antonym of "disabled", generalization of components by their container (or system), e.g., the relationship between ASL and its components, etc. Once the ontology is constructed considering the event-based semantic, thematic roles, lexical relation, we check the consistency of it via an inference engine (or reasoner), which is a software program that applies logical rules on a logical system, such as the ontology, to obtain new information.

Definition 4.2.2. The ontology is consistent if there exists an interpretation (or a model) M that satisfies the terminological assertions T and the concrete assertions A, that is, $M \models ax$, where: $ax \in T \cup A$, where T is a set of

terminological assertions, such as InParameter $\subset$ Parameter, and A is a set of facts (or concrete assertions).

4.3 Formal and Scalable Analysis of Simulink Models

A Simulink model consists of functional blocks, which are modeling elements to realize some functionlity, e.g., generate signal, delay a signal, mathematical operations over signals, etc., and lines connecting the blocks. The Simulink block can be either discrete or continuous depending on the block's execution behavior. If the block is executed periodically with a sample time t_s, it is descrete, otherwise, it is continuous if it has no sampling time, rather executes over finitely small (or minor) sampling times. The blocks can also be classified into atomic and composite blocks, where the atomic blocks, e.g., a delay block, is atomic schedulable entity that cannot be decomposed further. Whereas, the composite blocks, e.g., the Subsystem block, contains atomic blocks, and is used to create hierarchy, hence improving the visualization or to impose collective execution behavior over a set of atomic blocks.

4.3.1 Semantics of Simulink Model in Network of STA

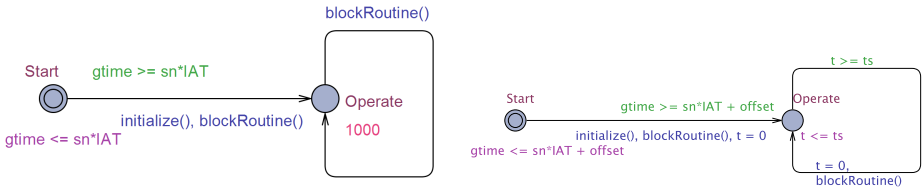
The semantics of a Simulink model is given as a network of stochastic timed automata, which is defined as a composition of independently executing automata (processes). The automata are labeled transition systems which denote the sematics of Simulink blocks, and the syntax of a block is captured as a tuple

Definition 4.3.1 (Simulink model)**.** A Simulink model is formally defined as a sequential composition of n Simulink blocks that communicate via shared variables, as follows:

$$S = B_1 \otimes B_2 \otimes B_3 \cdots \otimes B_n, \tag{4.1}$$

where: $B_1, B_2, \ldots, B_n$ are computational Simulink blocks and $\otimes$ is a composition operator.

The syntax of a Simulink block is given as a tuple $\langle s_n, V_{in}, V_{out}, V_D, \Delta, Init, blockRoutine \rangle$, where s_n is the execution order of the block, V_{in}, V_{out} and V_D are the set of input, output and state variables of the block. Δ are the set of sample hits, when block update functions are executed. The $Init$ and $blockRoutine$ functions performs intitialization of the block's state variables and executes the update functions of the block, respectively.



(a) Continuous. *(b)* Discrete.

Figure 4.4: STA transformation pattersn.

Definition 4.3.2 (Semantics of a Simulink block)**.** Assume B is a Simulink block, the semantics of B is a timed transition system $T_B = \langle q_0, Q, \mathcal{L}, \longrightarrow \rangle$, where

- $Q = \mathbb{R}^n$ is the state space given by the values of all output variables y at a given time instance $t \in \mathbb{R}_{\geq 0}$, for given input at time t
- $q_0 = y_0|_{t_0} = (y_0, t_0) \in Q$ is the initial state at $t_0 \in \mathbb{R}_{\geq 0}$
- $\mathcal{L} = \mathcal{L}_a \cup \mathcal{L}_t$ is the set of labels, with $\mathcal{L}_a$ the set of action labels $\{Init, blockRoutine\}$, and $\mathcal{L}_t$ the set of time labels $\{r * \delta, m * \delta\}$
- $\longrightarrow$ is the transition relation from $\longrightarrow \subseteq Q \times \mathcal{L}_a \times \mathcal{L}_t \times Q$,

where $r * \delta \in \mathbb{R}_{\geq 0}$ are the set of sample hits that occur every infinitesimal sample time r, to approximately model the continuous behavior of blocks, and $m * \delta$ are the sample hits that occur every sample time m, to model the behavior of discrete blocks. We refer the reader to Section 3 of our paper [25] for the detailed formalization discussion.

Transformation Patterns: In order to facilitate the transformation of the Simulink blocks into their equivalent transition systems, we propose two types of transformation patters for the continuous-time and discrete-time blocks. The transformation patterns are reusable and they conform to the semantics of the tuple introduced in Equation 4.1. Figure 4.4 shows our transformation patterns encoded in the input language of UPPAAL SMC.

Each pattern has its own execution mechanism. The execution of continuous-time patterns presented in Figure 4.4a proceeds according to an exponential distribution for unbounded delays, whereas the execution of the discrete-time pattern presented in Figure **??** proceeds according to the uniform distribution for time-bounded delays modeled via the invariant.

To check the correctness of the `blockRoutine` functions, which we implement in the subset of the C language used by UPPAAL SMC, we use pre-/post-condition verification using the Dafny [**?**] program verifier. A set of pre-conditions is used to describe the input, output and state variables prior to the execution of the `blockRoutine`. Given that the pre-condition

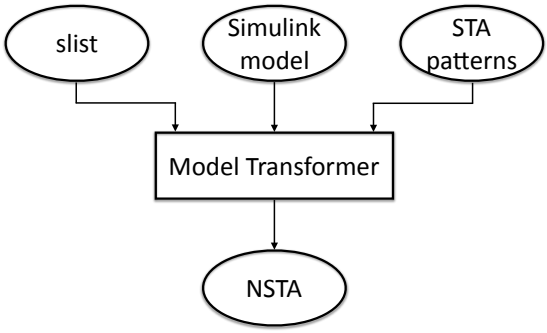


Figure 4.5: Simulink to NSTA Transformation

holds, after the execution of the blockRoutine, the set of post-conditions has to be established. We consider the `blockRoutine` to be correct if the specified set of postconditions is satisfied for all executions. For complex block routines that contain loops, we use loop invariants and termination conditions. The detailed description of the verification procedure for the `blockRoutine` using `Dafny` has been omitted for brevity. For full details, we refer the reader to our technical report [**?**].

Transformation process: The network of automata is automatically generated from the Simulink model by applying the following steps:

Step 1 The Simulink model is simulated, subsequently, the sorted-order list, which contains the execution order of the Simulink blocks, is extracted.

Step 2 In case the model is hierarchical, the execution order is flattened first, thus generating a flattened sorted-order list.

Step 3 The Simulink model is parsed, as a result, each block is translated into a stochastic timed automata via a transformation pattern, which takes its block routine and the flattened sorted-order list.

Step 4 In the transformation, the continous-time STA and the discrete-time STA patterns are applied to continuous-time and discrete-time blocks, respectively.

Step 4 the connections between blocks are translated into input-out variables of the automata

4.3.2 Validation on the Brake-by-wire System

To validate the proposed approach, we use a brake-by-wire use case that is obtained from Volvo Group Trucks Technology. The brake-by-wire system

controls the vehicle speed using sensors and actuators, e.g., by reading the pedal position, vehicle speed, etc., and employs an elctromecahnical control system to compute the proportional torque force that is applied on the wheels. Besides realizing the braking functionality, the model realizes anti-braking functionality to avoid (or minimize) skidding, which occurs when the slip rate of a wheel is greater than its friction coefficient. The anti-braking feature is activated whenever the vehicle speed is greater than 10km/hr.

The brake-by-wire Simulink model contains 320 blocks which is structured into a 4-level hierarchy, and consists of continuous and discrete blocks. Following the transformation, the model is first simulated in Simulink, subsequently, the sorted-order lists is extracted. Then, the model is transformed into a netwok of stochastic timed automata via the transformation patterns. Note: the timing behavior of the automata complies to the sorted order list, thus preserving the exectuiong semantics of the Simulink model.

4.4 Exact Software Allocation via ILP

Safety-critical systems are usually resource constrained, that is, the the execution hardware platform has limited computational resources and power/energy sources. Therefore, the system should be designed efficiently especially critical system resources such as power in order to accommodate complex safety-critical software functionality. In this section, we propose exact method of mapping software to hardware in order to optimize the total power consumption of a distributed safety-critical software via ILP.

We consider an AUTOSAR distributed safety-critical software that can be mapped to multiple computing nodes (or units), which are heterogenous computing systems with respect to power specification, failure rate and processing speed. The software has end-to-end timing and reliability requirements, therefore, the mapping must satisfy the requirements. In order to meet the reliability requirement, the mapping applys fault tolerance by replicating software components subsequently mapping the replicas on different computing units. Moreover, the task and messages that realize the safety-critical software functionality are scheduled using a fixed-priority preemptive scheduling policy and fixed-priority non-preemptive scheduling policy (over a CAN bus), respectively.

The AUTOSAR software functionality is implemented by Runnables that communicate over the virtual function bus (VFB), which abstracts the details of the run-time environment. We assume the runnables are triggered periodically and have multiple worst-case execution times as they can executed on processors that have different processing speed.

4.4.1 ILP Model Formulation

Consider that the mapping solution is represented by a vector of binary matrices $\mathbf{x} = \{\mathbf{x}_1 : 1,...,K\}$, where x_{ij}^krepresentes the mapping of the software component $q_{ij}^{A_k}$ to the computing node $n_j \in \mathcal{N}$.

$$\mathbf{x}^k = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1K}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2K}^k \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1}^k & x_{N2}^k & \cdots & x_{NK}^k \end{bmatrix} \tag{4.2}$$

The power consumption of a software appliation $\{c_i : i = 1,...,n_C\}$ that is mapped to computing units $N' \in N$ is calculated as the sum of the power consumption of each node $n_h \in N'$, thus $\mathcal{P}_{total}(x) = \sum_{n \in N'} \mathcal{P}(u_n(x))$, where $\mathcal{P}(u)$ is the power consumption of a computing node which linearly proportional to its utilization [43]. The utilization of a comuting node is the sum of the utilization of its constituent tasks that realize corresponding software components.

$$(u_1,...,u_K)(\mathbf{x}) = \sum_{k=1}^{K} \sum_{\tau \in T_{c_i}} x_{ij}^k * \frac{WCET_\tau}{P_\tau}, \tag{4.3}$$

where T_{c_i} is the set of tasks realizing the software component c_i, $WCET_\tau$ and P_τ are the worst-case execution time and period of the task τ.

4.4.1.1 Tasks Deadline Constraint

The mapping solution must satisfy the timing and reliability requirements imposed on the safety-critiacal application. We assume the taskset in each computing node is scheduled using a fixed-priority preemptive scheduler [58] and we check the schedulability via the classical worst-case response time analysis [4]. Since the time analysis model involves recursion which is difficult to model as a linear model. Therefore, we construct linear logical constraints ξ that corresponds to the valid tasksets that can potentially partitioned to computing nodes, and these tasksets are schedulable.

The logical system is constructed as follows: first we identify the possible partitions of the software components, $P = 2^C$, then we check the schedulability of each partition and catagorize them per node, i.e., $Y_j = \{p \in P | isSched(p, n_j)\}$, where *sched* is a boolean function that returns true if p is schedulable on n_j. A partition is identified by a sequence of 0s and 1s based on its costituent components, e.g., the id of the partition $\{c_1, c_4, c_5\} \subseteq \{c_1, c_2, c_3, c_4, c_5\}$ is $10011 = 19$. So, given the mapping $\mathbf{x}$, the

partition id that is mapped to the computing node n_j can be computed as

$$g_j(\mathbf{x}) = \sum_{i=1}^{N} 10^{N-i} \max_{1 \leq k < K} x_{ij}^{k} \tag{4.4}$$

where the $\max_{1 \leq k < K}$ function returns 1 if at least one component replica is mapped n_j, otherwise returns 0, indicating no replica is mapped to that node. Thus, each node must have a valid partition that is schedulable as indicated in the logical constraint,

$$\bigvee_{p \in Y_j} g_j(\mathbf{x}) = id(p), \tag{4.5}$$

where $id(p)$ is a predicate that returns the id of the partition p.

4.4.1.2 End-to-end Timing Constraint

The age delay of a chain is computed according to the semantics discussed in [21][51]. The response-time logical assertsion enure that every task in the distributed system are schedulable for a mapping $\mathbf{x}$. Therefore, we can safely check if the age delay of each chain satisfy its end-to-end timing requirement without the need to check the chain's constituent tasks are schedulable or not. Similar to the previous approach, we construct logical constraints η that correspond to the valid set of mappings of chains on the network of computing nodes. Given $\Gamma = \{\Gamma_i : i = 1, ..., N\}$ of chains, the set of possible mappings of each chain Γ_i on a set of $\mathcal{N}^{\Gamma}$. Thus the set of valid mappings of each chain satisfy the end-to-end timing requirement is $Z_i = \{\gamma \in \mathcal{N}^{\Gamma} | ageDelay(\gamma) < EE_{\gamma}$.

4.4.1.3 Software Application Reliability Constraint

The reliability constraint refers to the probability that the application functions by the time t, i.e $[0,t]$ [28]. We assume that the reliability requirement of a software application is 0.99999999 over a 2-year operation time, which is to say that the safety-critical system almost should not fail within the stated period. Furthermore, we assume that the mean-time to failure of the computing nodes is given as 10^6 hours (0.0000001 per hour), which is the usual practice in many functional safety design.

Unless the the software application is replicated on one or more computing nodes, the reliability requirement is usually unattainable. So, the allocation strategy is basically to replicate sufficient software components to meet the requirement at the same time it should not exceed the limit so that the application consumes less power. However, the reliability calculation is not trivial as the replication results in functional interdependency of the comuting nodes, in this case, the series-parallel method does not apply. For this reason, we propose an exact method based on state enumeration [**?**], which basically enumerates all the possible configurations of the nodes *PS*.

Subsequently, the reliability of the application becomes the total probability of the configurations that enable the functioning of the application [43].

Definition 4.4.1 (Software Application Failure)**.** A software application fails in the configuration $s \in \xi$ if there exists a component type c_i where all of its replicas Q_i *fail*, otherwise, it functions, as shown in Equation (4.6). The component replica $q_i, j \in Q_i$ of type c_i fails if n_h fails, that is $s_h = 0$.

$$f_s(x) = \left\lfloor \frac{\sum_i f_{c_i}(x,s)}{N} \right\rfloor = \begin{cases} 1 & \text{if } application \text{ functions} \\ 0 & \text{if } application \text{ fails} \end{cases} \tag{4.6}$$

Thus, the reliability of the software allocation is

$$Reliability(\mathbf{x}) = \sum_{s \in PS} f_s(\mathbf{x}) * p_s, \tag{4.7}$$

where p_s is the probability that the computing nodes attain a configuration s which is computed as $p_s = \prod_{m \in M} \left[(z_m * \lambda_m) + (1 - z_m) * (1 - \lambda_m) \right]$, where $z_m \in \{0,1\}$ indicates if the node fails (i.e., 0) or functions (i.e., 1), λ_m is the failure rate of the node m and $1 - \lambda_m$ is the opposite of failure rate, i.e., that is the rate that it does not fail.

4.4.1.4 ILP Problem

The goal of the ILP optimization is to minimize the total power consumption, which is the objective function, and the constraints Equation (4.9), (4.10) and (4.11)ensure that the optimization satisfiest the tasks deadine, end-to-end timing and reliability constraints, respectively.

$$\min_{\mathbf{x} \in X} \mathcal{P}_{total}(\mathbf{x}) \qquad \text{Subjected to:} \tag{4.8}$$

$$\bigvee_{p \in Y_j} g_j(\mathbf{x}) = id(p) \qquad \text{for all } j = 1,..,|\mathcal{N}| \tag{4.9}$$

$$\bigvee_{\gamma \in Z_j} h_j(\mathbf{x}) = id(\gamma) \qquad \text{for all } j = 1,..,|\Gamma| \tag{4.10}$$

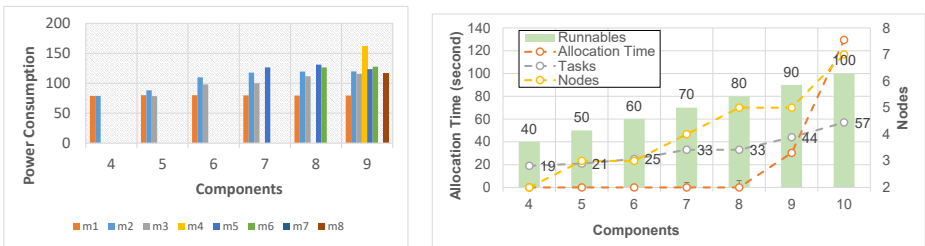
$$\sum_{s \in PS} f_s(\mathbf{x}) * p_s \leq \mathsf{RL} \qquad , \tag{4.11}$$

4.4.2 Tool Support and Validation on Automotive Benchmark

Our proposed ILP approach is validated on different size of software applications, i.e., applications with variable number of sotware components, cause-effect chains and degree of replication. The applications are synthesized according to the automotive benchmark proposed by Kramer et al. [37], which discusses a typical complexity of an AUTOSAR engine

management system in terms of, e.g., the number and execution time of runnables, and the number and activation patterns of cause-effect chains.

We prepared 6 software applications and an execution platform that consists 8 nodes. The smallest application consist of 4 components and the largest 10 comoponents. Furthermore, the cause-effect chains varies from 10 to 60 and activation patters ranges from 2 to 4. Figure 4.6 shows the result of the optization from the CPLEX solver, that is, the power consumption and the computation time, respectively. In general, the optimization maps components to nodes with lower power specification and higher processor speed, however, this is not always the case since the end-to-end timing and reliability constraints can dictate the selection of different nodes, which consume more power as a result. Figure 4.6b shows a slow increase of computation time from until the application with 8 number of components, which took 6.06 sec, and rapidly increased to 30.3 sec and 129.4 sec allocating application with 9 and 10 components, respectively. The optimization took extremely large amount of time for applications that consist of more than 10 components, and more than 15 component the waiting time was intractable, thus the opotimization was interrupted manually.



(a) Power consumption of nodes.

(b) Allocation times software applications.

Figure 4.6: ILP optimization of different sizes of software applications based on the number of software components, cause-effect chains, computing nodes.

4.5 Scalable Software Allocation via Hybrid PSO

In the previous contribution, we observe that the ILP approch does not scale well to the large software allocation problems. To address this problem, in this section, we propose approximation algorithms based on metahuristics, which is "high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms". Meta-heuristic algorithms does not gurantee optimality of solutions, neverthless, the solutions can be deemed satisfactory. In the case of minimizing power consumption, what is more appealing to system designers is usually the benefit attained as a result of the power consumption reduction,

e.g., accommodting more and more software applications, improving battery-life, rather than optimality. In this spirit, metahuristics can be useful to solve complex and large optimization problems, yet in reasonable time, as compared to exact methods, e.g., branch-and-bound, dynamic programming.

4.5.1 Hybrid Particle-swarm Optimization

The canonical PSO technique uses the constriction factors to balance exploitation and exploration of the search space to get closer to the global optima, hence improving solution quality. Nevertheless, it still suffers from premature convergence or local minima especially when applied on complex and large problems [**?**]. Its hybridization is proven to perform better in many cases [**?**]. In particular, it is shown to perform better in the tasks assignment problem, that is when hybridized with, e.g., the genetic algorithm [**?**], the hill-climbing [**?**], simulated annealing [**?**], differential evolution [**?**]. As compared to the hybridization with genetic, the hybridization with hill-climbing HCPSO is shown to perform better by Yin et al. [**?**] for the tasks allocation problem to maximize reliability of distributed systems.

In this work, we apply HCPSO to the problem at hand, and to tackle its stagnation when applied to large problems. Moreover, we hybridize PSO with the differential evolution technique, DEPSO, to improve diversification by applying the mutation and cross-over operators of the differential evolution. Algorithm 1 show the pseudocode of the hybrid PSO. Line 3 and 4 compute the personal best and the swarm best solutions, respectively. For each particle in the swarm, the velocity and position is computed in Lines 5-8. Lines 9-13 apply the hybridization based on the choice of the algorithm, i.e., DE, HCPSO and SHPSO

intermittently, i.e., whenever the interval criterion condition is met.

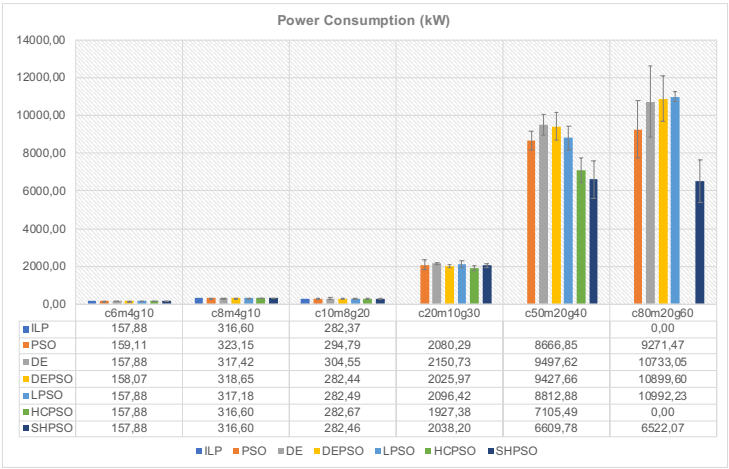
```
   input : PSOparameters, DEparameters
   output: Software allocation solution sBest.x
 1 Particles P ← initPSO();
 2 while termination criteria do
 3 |   p_bst ←ComputePersonalBest(P);
 4 |   z ←ComputeSwarmBest(P);
 5 |   foreach p ∈ P do
 6 |   |   computeParticleVelocity(p) according to
   |   |    Equation (2.9);
 7 |   |   computeParticlePosition(p) according to
   |   |    Equation (2.10);
 8 |   end
 9 |   if interval criteria then
10 |   |   P ← optimizeUsingDE(P);
11 |   |   // P ← optimizeUsingHC(P)
12 |   |   // P ← optimizeUsingSHC(P)
13 |   end
14 end
```

Algorithm 1: Hybrid PSO Pseudocode.

4.5.2 Validation on the Bosch EMS Benchmark

In this section, we evaluate our proposed hybrid PSO algorithms for the allocation of software applications to heterogeneous computing units, which conform to the system model presented in Section **??**. The algorithms are evaluated against different specifications of automotive software applications and execution platforms with regard to effectiveness, stability and scalability. The software-application specifications consist of the number of software components c, runnables r, tasks t and cause-effect chains g. The specifications are synthesized from the automotive benchmark proposed by Kramel et al. [37]. The benchmark indicates a strong correlation between runnables and cause-effect chains in terms of timing and activation patterns. It shows the timing specifications of runnables and their shares in an engine management system. Moreover, it shows the activation patterns of cause-effect chains, the runnables per activation and their shares in the system. The engine management system is one of the most complex automotive systems in the vehicular electrical/electronic execution platform.

Based on our experience in the automotive industry, the benchmark results are extrapolated to characterize different classes of automotive software application specifications, that is by varying the parameters related to the software components, runnables, and cause-effect chains. The different



	c6m4g10	c8m4g10	c10m8g20	c20m10g30	c50m20g40	c80m20g60
ILP	157,88	316,60	282,37			0,00
PSO	159,11	323,15	294,79	2080,29	8666,85	9271,47
DE	157,88	317,42	304,55	2150,73	9497,62	10733,05
DEPSO	158,07	318,65	282,44	2025,97	9427,66	10899,60
LPSO	157,88	317,18	282,49	2096,42	8812,88	10992,23
HCPSO	157,88	316,60	282,67	1927,38	7105,49	0,00
SHPSO	157,88	316,60	282,46	2038,20	6609,78	6522,07

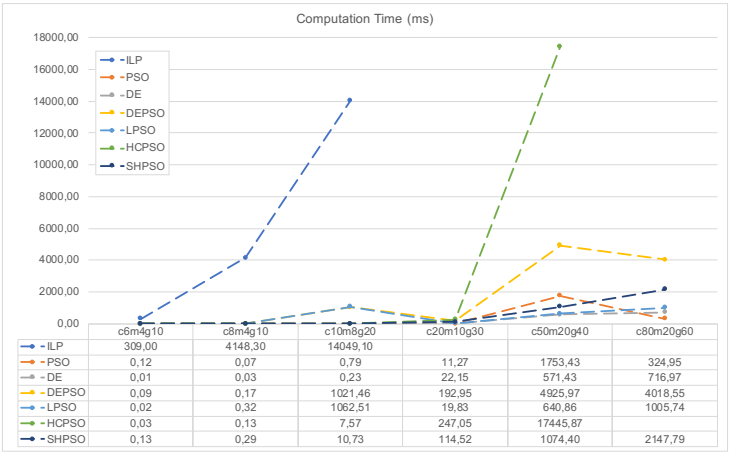
Figure 4.7: (Near) Optimal Power Consumption of the Different Software Allocation Problems.

classes of specifications range from Spec-I to Spec-V as shown in Table **??**. Likewise, the specifications for an execution platform consist of the processor speed, power specifications and failure rates of computing units.

Result: We conducted two experiments: i) the first experiment is designed to compare the performance such as the convergence time, computation time, optimality (or quality of solutions) and stability of the meta-heuristic algorithms used in this paper, ii) the second experiment is designed to evaluate the overhead of increasing replication on the optimization especially due to the computation of end-to-end delays, and also to evaluate the effect of the approximation algorithm proposed to reduce the overhead.

In the first experiment, as illustrated in Figure 4.7, we show that ILPreturned optimal solutions in the first three problems, likewise all except PSO and DEPSO. However, as the problem size inceases, the computation time became exponential as shown in Figure 4.8 and did not return solutions. In contrast, the hill-climbing-PSO hybrid performed the best in all last three problems except HCPSO failed to return near optimal solutions to the largest problem $c_{80}g_{20}n_{60}$. The result also shows that the stochastic version of the hill-climbing-PSO hybrid SHPSO scales better the over quality of the solution can deemed acceptable as compared to the ILP in the first three problems and HCPSO in the next two problems.

Due to the replication of software components to the reliability goals of the softwar applications, the overhead of computing the chains delay is exponential. Figure x shows the results of the power consumption and the computation time after afer applying the approximation algorithm that is introduced the overhead. Specifically, the results show $61\% - 81\%$ computation time improvement over the exact method while facing quality



	c6m4g10	c8m4g10	c10m8g20	c20m10g30	c50m20g40	c80m20g60
ILP	309,00	4148,30	14049,10			
PSO	0,12	0,07	0,79	11,27	1753,43	324,95
DE	0,01	0,03	0,23	22,15	571,43	716,97
DEPSO	0,09	0,17	1021,46	192,95	4925,97	4018,55
LPSO	0,02	0,32	1062,51	19,83	640,86	1005,74
HCPSO	0,03	0,13	7,57	247,05	17445,87	
SHPSO	0,13	0,29	10,73	114,52	1074,40	2147,79

Figure 4.8: Computation time of the various algorithms for solving different instances of the software allocation problem.

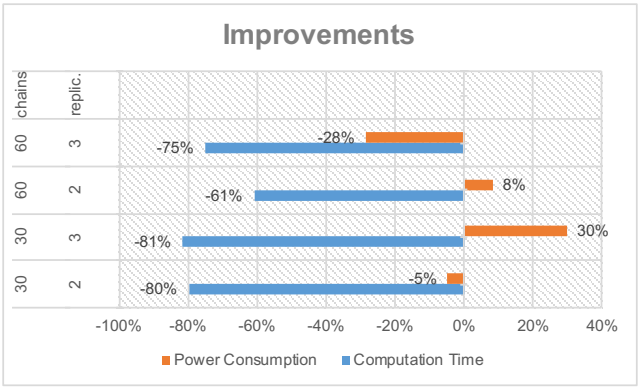


Figure 4.9: Effect of approximate algorithm over delay calculations with replication.

degradation only for samples $g_{30}d_3$ and $g_{60}d_2$. The improvements are in seconds, which implies for a single usage (or run) of the meta-heuristic optimization algorithms, it is not significant. However, considering practical systems design process, which requires several iterations, the commutative effect of the algorithms can negatively impact the responsiveness to engineers. Thus, the improvements can be in trade-off with optimality of the solutions.

4.6 Paper Contributions

Tables 4.4 shows the peer-reiewed papers and their contribution to the thesis.

Paper	Sec4.1	Sec4.2	Sec4.3	Sec4.4	Sec4.5
A	×				
B	×	×			
C		×			
D			×		
E				×	
F				×	×

Table 4.4: *List of the peer-reviewed papers and their contribution to the thesis.*

Paper A

ReSA: An ontology-based requirement specification language tailored to automotive systems. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar.*In the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)(pp. 1-10). IEEE.*

Abstract: *Automotive systems are developed using multi-leveled architectural abstractions in an attempt to manage the increasing complexity and criticality of automotive functions. Consequently, well-structured and unambiguously specified requirements are needed on all levels of abstraction, in order to enable early detection of possible design errors. However, automotive industry often relies on requirements specified in ambiguous natural language, sometimes in large and incomprehensible documents. Semi-formal requirements specification approaches (e.g., requirement boilerplates, pattern-based specifications, etc.) aim to reduce requirements ambiguity, without altering their readability and expressiveness. Nevertheless, such approaches do not offer support for specifying requirements in terms of multi-leveled architectural concepts, nor do they provide means for early-stage rigorous analysis of the specified requirements. In this paper, we propose a language, called ReSA, which allows requirements specification at various levels of abstraction, modeled in the architectural language of EAST-ADL. ReSA uses an automotive systems' ontology that offers typing and syntactic axioms for the specification. Besides enforcing structure and more rigor in specifying requirements, our approach enables checking refinement as well as consistency of requirements, by proving ordinary boolean implications. To illustrate ReSA's applicability, we show how to specify some requirements of the Adjustable Speed Limiter, which is a complex, safety-critical Volvo Trucks user function.*

Personal Contributions: I was the main driver of the paper. I developed the ReSA language including its syntax and semantics, and Cristina Seceleanu proposed a consistency analysis technique besides giving useful comments

and ideas on the design of the language. Oscar Ljungkrantz provided useful materials from VGTT that were eventually analyzed for the language development, and gave feedback on the language design and implementation from an industrial viewpoint.
Status: Published

Paper B

ReSA tool: Structured requirements specification and SAT-based consistency-checking. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar. *In the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS)(pp. 1737-1746). IEEE.*

Abstract: *Most industrial embedded systems requirements are specified in natural language, hence they can sometimes be ambiguous and error-prone. Moreover, employing an early-stage model-based incremental system development using multiple levels of abstraction, for instance via architectural languages such as EAST-ADL, calls for different granularity requirements specifications described with abstraction-specific concepts that reflect the respective abstraction level effectively. In this paper, we propose a toolchain for structured requirements specification in the ReSA language, which scales to multiple EAST-ADL levels of abstraction. Furthermore, we introduce a consistency function that is seamlessly integrated into the specification toolchain, for the automatic analysis of requirements logical consistency prior to their temporal logic formalization for full formal verification. The consistency check subsumes two parts: (i) transforming ReSA requirements specification into boolean expressions, and (ii) checking the consistency of the resulting boolean expressions by solving the satisfiability of their conjunction with the Z3 SMT solver. For validation, we apply the ReSA toolchain on an industrial vehicle speed control system, namely the Adjustable Speed Limiter.*

Personal Contributions: I was the main driver of the paper. I developed the ReSA toolchain that consists of the editor and the consistency checker including the integration with the Z3 SAT solver in the backend. Cristina Seceleanu formulated the consistency checking and together with Oscar Ljungkrantz, they contributed to the paper with useful comments and ideas.
Status: Published

Paper C

Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic. Nesredin Mahmud, Cristina Seceleanu and Ljungkrantz Oscar. *In the International Conference on Software Engineering and Formal Methods (pp. 332-348). Springer, Cham.* Acceptance

rate: 26%.

Abstract: *Due to the increasing complexity of embedded systems, early detection of software/hardware errors has become desirable. In this context, effective yet flexible specification methods that support rigorous analysis of embedded systems requirements are needed. Current specification methods such as pattern-based, boilerplates normally lack meta-models for extensibility and flexibility. In contrast, formal specification languages, like temporal logic, Z, etc., enable rigorous analysis, however, they usually are too mathematical and difficult to comprehend by average software engineers. In this paper, we propose a specification representation of requirements, which considers thematic roles and domain knowledge, enabling deep semantic analysis. The specification is complemented by our constrained natural language specification framework, ReSA, which acts as the interface to the representation. The representation that we propose is encoded in description logic, which is a decidable and computationally-tractable ontology language. By employing the ontology reasoner, Hermit, we check for consistency and completeness of requirements. Moreover, we propose an automatic transformation of the ontology-based specifications into Timed Computation Tree Logic formulas, to be used further in model checking embedded systems.*

Personal Contributions: I was the main driver of the language. I developed the ReSA language semantics using event-base approach, which is encoded in description logic. Cristina Seceleanu and Ljungkrantz Oscar provided with useful ideas and comments.
Status: Published

Paper D

SIMPPAAL - A Framework For Statistical Model Checking of Industrial Simulink Models. Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz and Henrik Lönn. *Submitted to ACM Transaction on Software Engineering and Methodology (TOSEM). ACM Journals.*

Abstract: *The evolution of automotive systems has been rapid. Nowadays, electronic brains control dozens of functions in vehicles, like braking, cruising, etc. Model-based design approaches, in environments such as MATLAB Simulink, seem to help in addressing the ever-increasing need to enhance quality, and manage complexity, by supporting functional design from a set of block libraries, which can be simulated and analyzed for hidden errors, but also used for code generation. For this reason, providing assurance that Simulink models fulfill given functional and timing requirements is desirable. In this paper, we propose a pattern-based,*

execution-order preserving automatic transformation of atomic and composite Simulink blocks into stochastic timed automata that can then be formally analyzed with Uppaal Statistical Model Checker (Uppaal SMC). To enable this, we first define the formal syntax and semantics of Simulink blocks and their composition, and show that the transformation is provably correct for a certain class of Simulink models. Our method is supported by the SIMPPAAL tool, which we introduce and apply on two industrial Simulink models, a prototype called the Brake-by-Wire and an operational Adjustable Speed Limiter system. This work enables the formal analysis of industrial Simulink models, by automatically generating stochastic timed automata counterparts.

Personal Contributions: The three co-authors contributed equally to writing the paper. Technically, I equally contributed with proposing the pattern-based semantics of Simulink blocks, together with Predrag Filipovikj. I introduced a mechanism to enforce the execution order of the blocks using inter-arrival times. Predrag implemented the flattening algorithm and the tool for the automatic transformation of Simulink models into a network of timed automata with stochastic semantics. Raluca Marinescu contributed with analyzing the BBW system, Cristina Seceleanu contributed with defining the methodology, and with useful ideas and comments. Guillermo Rodriguez-Navas wrote the related work section. The industrial coauthors provided the use cases and commented on the final draft.
Status: Under Review

Paper E

Power-aware Allocation of Fault-tolerant Multirate AUTOSAR Applications. Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Faragardi,Saad Mubeen and Cristina Seceleanu. *In the 25th Asia-Pacific Software Engineering Conference (APSEC'18). IEEE. .*

Abstract: *The growing complexity of automotive functionality has attracted revolutionary computing architectures such as mixed-criticality design, which enables effective consolidation of software applications with different criticality on a shared execution platform. Mixed-critical design that is required to satisfy end-to-end timing and reliability specifications should consider power-efficient software design in order to accommodate more and more functionality. Due to the recursive and exhaustive nature of the real-time and reliability analysis, exact methods, e.g., branch and bound, dynamic programming, are prohibitively expensive. We propose hybrid particle-swarm optimization algorithms based on differential evolution and hill-climbing algorithms to minimize power consumption of the safety-critical software, which have end-to-end timing and reliability requirements, on a network of heterogeneous computing units. The optimization approach*

employs fault tolerance to maximize reliability of the software applications subsequently meet the reliability requirements. Our proposed integrated software-allocation approach is evaluated using a range of synthetic software applications based a real-world automotive benchmark. The evaluation makes comparative analysis of the differential evolution, particle-swarm optimization, integer-linear programming and hybrid particle-swarm optimization algorithms. The results show that the hybrid algorithms based on the hill-climbing algorithms outperform the rest of the meta-heuristic algorithms, in particular, the stochastic version of the hill-climbing algorithm scales well in large software allocation optimization problems while its overall optimality performance can be deemed acceptable.

My Contributions: I am the main driver of the paper. I developed the system model (including the power consumption, timing, reliability models) and further refined by the co-authors. Hamid Faragardi and I developed the ILP model, and I implemented the ILP problem (including the system model) and collected experimental results. The co-authors gave useful ideas and comments on respected parts of the paper: Guillermo Rodriguez-Navas on reliability modeling, Hamid Faragardi on optimization and related work, Saad Mubeen on the timing analysis, and Cristina Seceleanu gave comments and ideas on the main contributions of the paper, including on the optimization objective and constraints.
Status: Published

Paper F

Optimized Allocation of Fault-tolerant Embedded Software with End-to-end Timing Constraints

Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Faragardi,Saad Mubeen and Cristina Seceleanu. Optimized Allocation of Fault-tolerant EmbeddedSoftware with End-to-end Timing Constraints. *Mälardalen Real-time Research Center Technical Report (MRTC)*. Submitted to Elsevier JSA Journal.

Abstract: *The growing complexity of automotive functionality has attracted revolutionary computing architectures such as mixed-criticality design, which enables effective consolidation of software applications with different criticality on a shared execution platform. Mixed-critical design that is required to satisfy end-to-end timing and reliability specifications should consider power-efficient software design in order to accommodate more and more functionality. Due to the recursive and exhaustive nature of the real-time and reliability analysis, exact methods, e.g., branch and bound, dynamic programming, are prohibitively expensive. We propose hybrid particle-swarm optimization algorithms based on differential evolution and*

hill-climbing algorithms to minimize power consumption of the safety-critical software, which have end-to-end timing and reliability requirements, on a network of heterogeneous computing units. The optimization approach employs fault tolerance to maximize reliability of the software applications subsequently meet the reliability requirements. Our proposed integrated software-allocation approach is evaluated using a range of synthetic software applications based a real-world automotive benchmark. The evaluation makes comparative analysis of the differential evolution, particle-swarm optimization, integer-linear programming and hybrid particle-swarm optimization algorithms. The results show that the hybrid algorithms based on the hill-climbing algorithms outperform the rest of the meta-heuristic algorithms, in particular, the stochastic version of the hill-climbing algorithm scales well in large software allocation optimization problems while its overall optimality performance can be deemed acceptable.

Personal Contributions: I am the main driver of the paper. I developed the system model, the metaheuristic formulation, implemented the problem in Java, and collected and analyzed the experimental results. The co-authors gave writing update, useful ideas and comments on respected parts of the paper: Guillermo Rodriguez-Navas on reliability modeling, Hamid Faragardi on optimization, Saad Mubeen on the timing analysis, and Cristina Seceleanu gave idea on the main objective and constraints.
Status: Submitted to Journal of System Architecture (JSA), Elsevier Journals.

5. Research Method

Research methods, according to Jane et al. [5], are approaches, procedures and guidelines that are applied to conduct research, e.g., observation, interview, prototyping, experiment, etc. In this research, two main factors have driven the selection of research methods: i) the fact that the research is applied in industry (or involves industry-academia collaboration), which means the research results as well as the process should consider the interest and the nature of the industry, and ii) seamless integration of formal methods into existing engineering methods and practices with minimal cost, which requires careful consideration of existing engineering methods, guidelines, tools and capabilities. To summarize the main problems raised by the industry-academia collaboration: i) the research goals should target existing problems in the industry; ii) existing methods and tools should be leveraged; iii) proposed ideas, methods and tools should be agreed by both academia and industry teams before their design, implementation, validation, and integration to ensure usefulness; iv) furthermore, it is imperative that the proposed tools and methods are engineering-friendly, which means the research team is required to communicate with the actual users to capture the feel of the proposed methods and tools.

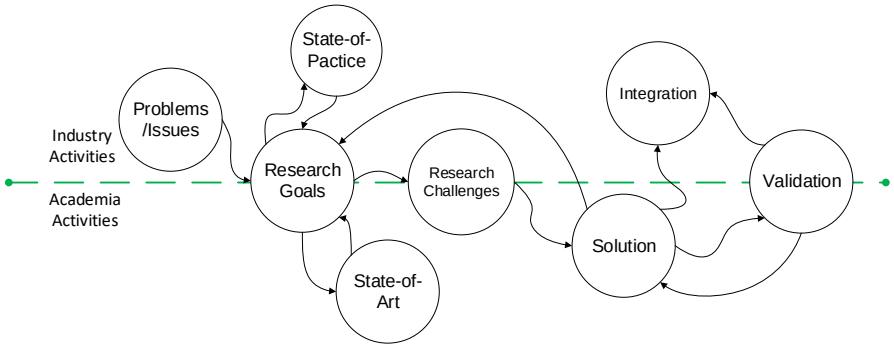


Figure 5.1: Research Process.

Figure 5.1 illustrates the research process employed in the thesis, which is an adaptation of the technology-transfer process proposed by Tony et al. [29]. Our main research activities has involved identification of research goals and research challenges, which are followed by solution proposals and

their implementations. Subsequently, the solutions are validated on industrial use cases, and are also integrated seamlessly into industrial tool chains. In order to conduct these activities, several quantitative and qualitative research methods are blended in order to gather, analyze and interpret, respectively, quantitative and qualitative data via what is known as *hybrid (or mixed) research* [13]. The benefit of applying mixed research methods is mainly in the triangulation of research outcomes through various research methodical approaches. In fact, this can be better explained by the requirements specification research problem, which has accommodated empirical research via interview, observation to collect quantitative data that has allowed us to understand the current practices and needs of VGTT. Subsequently, we have proposed the ReSA framework, which have been validated through quantitative methods including statistical analysis and questionnaire on its effectiveness and usability[1].

During implementations of the proposed solutions, we have applied a prototyping method [11], which has enabled incremental development of the solutions, before introducing grand progress in subsequent development phases, via concept modeling, implementation, demonstration and revision. This has been the case during the implementation of the ReSA framework [41, 42] as well as the SIMPPAAL framework [23]. The prototyping has involved concept development, and experimentation with toy examples and use cases to internally validate the implementation solutions, before validation by practitioners. Of course, the implementation of the research process has not been straightforward; on the contrary, similar to other research activities, it has accommodated several iterative, cyclic activities to clarify research goals and update solutions based on knowledge gained via literature study and feedback from industry. Besides, the industrial flavor of the research has required persistent synchronization meetings through discussions and workshops in order to update to the state-of-the-art and state-of-the-practice methods and tools.

[1]Work in progress - validation of the ReSA toolchain by practitioners, at VGTT.

6. Related Work

In this section are discussed the related work on specification and analysis of requirements, allocation of software architecture, and formal analysis of behavioral models, designed in Simulink.

6.1 Requirements Specification and Analysis

Embedded system requirements are captured in different representations, e.g., textual, tabular, graphical, etc. The textual representation, which is the scope of the analysis, can be conveniently classified into two classes: i) controlled natural language (CNL), ii) template-based methods. The syntax and semantics of the CNLs are similar to natural language except that the lexicon and the syntax are restricted for different reasons, of which improving comprehensibility of the text and formal representations to support rigour analysis of the text are prominent. The ReSA language is designed to improve comprehensibility of requirements specifications as well as to be computer-processable. There are many computer-processable CNL in literature [38], e.g., Attempto Controlled English (ACE) [27], Processable English (PENG) [57], etc. Similar to most computer-processable languages, ReSA has limited syntactic constructions, and allows knowledge-representations, in contrast though, our language is catered for embedded systems and therefore it uses concepts as well as semantic rules that are domain-specific to embedded systems. Similar to PENG, its implementation supports look-ahead in order to enable predictive and guided specification.

The template-based methods, in particular requirements boilerplate uses templates (or boilerplates) which are reusable, recurrent patterns to specify requirements, e.g., CESAR boilerplates [20], RAT, etc. The main drawback of existing requirements boilerplates are: i) the templates are usually too limited therefore not expressive enough, ii) it is not easy to find the appropriate boilerplate during specification. In this regard, ReSA extends boilerplates with a meta-model that guides for plausible instantiation of boilerplates.

6.2 Formal Analysis of Simulink Models

Several research endeavors have tackled the problem of formally analyzing Simulink models in order to gain better insight into the design of Simulink model, and they mainly differ in the aspects (or coverage) of the Simulink language that has been targeted for analysis and the formalisms applied. Besides to the scalability of the techniques, their robustness to formally analyze various types of Simulink models is also crucial especially for the proposed methods to be useful in industry. In this related work on formal analysis of Simulink models, existing formalisms and their applicability in industry are discussed, and also compared to our solutions.

Simulink already suppors formal analysis of Simulink models via its Simulink Design Verifier (SDV) product[1]. Though, there is limited resource to investigate about the pros and cons of the product, the study by Nellen et al. [53] indicates some limitations on its capability such as inconclusiveness on the verification results, lack of support to verify timed properties. The latter pros is also mentioned by Florian et al. [40] in the comparison against SPIN. For brevity, other approaches are classified into three categories, approaches that use Simulink traces, contract/theorem and model-to-model transformation.

The PlasmaLab proposed by Nikolaos et al. [34] transforms Simulink sample traces into statistical models, which are eventually analyzed by their statistical mode checker. Albeit the checker is assisted by an algorithm that determines sufficiency of the sample traces, it is unclear how it works. Unlike many approaches, PlasmaLab can analyze any Simulink models as long as the simulation results sample trances, which is the main advantage. Ferrante et al. [30] use contract-based theory in order to lift the block specification, and rely on a combination of SAT solvers and the NuSMV model checker for analysis. Hocking et al. [22] use the PVS specification language for writing the specification, and rely on the PVS theorem prover for analysis. A limitation of this strategy is that both steps still require much user interaction, so it is error-prone and requires certain understanding of the formal analysis engines, which is not common among embedded systems engineers.

The model-to-model transformation approach basically transforms the Simulink model into a formal model that can be checked via model checking, e.g., for reachability properties. Bernat et al. [49] proposed transformation Simulink models that consist only discrete blocks, which are subsequently checked via the DiViNE model checker. The research endeavors proposed transformation only StateFlow/Simulink into timed and hybrid automata. The discussed model-to-model approaches are limited in scalability due the use of exact model checking. In contrast, our approach uses statistical model checking, and thus scales better albeit is not exhaustive. However, by collecting sufficient sample traces and consequently acceptable

[1] https://se.mathworks.com/products/sldesignverifier.html

confidence value in the statistical analysis, the later challenge can be tackled. Furthermore, our approach supports transformation of any discrete and continuous blocks, and also blocks that are custom and StateFlow. The latter capability is achieved since our transformation abstracts the internal implementation of the blocks.

6.3 Software-to-Hardware Allocation

Different allocation schemes deliver different system performance and therefore efficient software allocation is crucial. Ernest Wozniak et al. [61] proposed a synthesis mechanism for an AUTOSAR software application that can executed over multiple nodes, with the objective fulfilling timing requirements. In contrast, we consider power consumption and reliability requirements besides timing. Similarly, Salah Saidi et al. [56] proposed an ILP based approach for allocation of an AUTOSAR application on a multi-core framework in order to reduce the overhead of inter-process communication while we consider a multi-nodes platform. Ivan Svogor et al. [59] proposed a generic approach of identifying resource constraints and a way of handling different measurement units with Analytic Hierarchy Process (AHP) in order to allocate a component-based software application on a heterogeneous platform. However, the resource constraints are trivialized, e.g., end-to-end delay calculations, which require timing specifications and activation patterns of tasks. As opposed to the previously mentioned related work, we considered a system model with a multi-rate software application, which basically impose complex timing analysis due to complex timed paths from the source to the sink of communication signals [51]. On a different work, there are research focusing on power and energy consumption in real-time distributed systems which are employ dynamic voltage scaling [3] and task consolidation by minimizing computational nodes [19] [16].

7. Conclusions and Future Work

Improving functional safety and quality of embedded systems is crucial due to the catastrophic consequences of their failure. Besides their applications in ragged environment, operating for a along time without interruption and the fact that such system are usually resource constrained, call for a systematic development approach. In this thesis, we have proposed several formal techniques for improving functional safety and quality of embedded systems, including requirements specifications, software system architecture, and software behavior, modelled in Simulink.

We proposed a domain-specific language for the specification of embedded systems requirements called ReSA. The language resembles the natural language English in syntax and semantics. However, its syntax is constrained in order to improve comprehensibility of the specifications. The language has formal semantics in Boolean and description logic, which has enabled for superficial and deep analysis of the requirements, e.g., consistency checking. The ReSA toolchain contains a ReSA editor that supports content-completion to guide requirements specification, and the specifications can be checked for consistency via the Z3 SMT solver. The language is validated for its expressively on a set of 300 industrial requirements, and has proven to express about 90% of the requirements.

We have also provided a mechanism to preserve timing and reliability requirements of multirate software applications during software allocation, while minimizing power consumption via Integer Linear Programming, ILP (exact) and Particle Optimization, PSO (meta-heuristic) methods. The ILP method is shown to be applicable to allocation of small and medium of software applications, whereas the PSO has shown to scale well for allocation of large software applications.

Furthermore, we have introduced an automated pattern-based, order-preserving method for transforming behavioral embedded systems models in Simulink into networks of stochastic timed automata analyzable by UPPPAAL SMC. The method is implemented in our tool SIMPPAAL that we have integrated with ReSA and validated on the BBW industrial prototype.

The ReSA language is a descriptive and intuitive language as it resembles natural language. By extending its constructs to encompass design constraints, it should be possible and in fact beneficial to synthesize a high-level architecture, using correct-by-construction. Furthermore, by raising detailed hardware platform specifications in to the system design,

e.g., memory, CPU, power specification, effective software to hardware allocation can be achieved. The proposed formal analysis of Simulink models can be generalized to other language that comply to a data-flow programming paradigms, e.g., LabView.

Bibliography

[1] ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, 12 2011.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. *Kybernetes*, 2010.

[3] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware Scheduling for Real-time Systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):7, 2016.

[4] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings - Real-Time Systems Symposium*, 2011.

[5] Bruce L (Bruce Lawrence) Berg and 1962-Author Lune Howard. *Qualitative research methods for the social sciences*. Boston Pearson, eighth edi edition, 2012.

[6] Mehul Bhatt and Christian Freksa. Spatial Computing for Design—an Artificial Intelligence Perspective. In John S Gero, editor, *Studying Visual and Spatial Reasoning for Design Creativity*, pages 109–127, Dordrecht, 2015. Springer Netherlands.

[7] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Handbook of Satisfiability. *New York*, 185(3):980, 2009.

[8] Willem Nico Borst and W N Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente, Netherlands, 1997.

[9] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. *Electronic Proceedings in Theoretical Computer Science*, 2012.

[10] Giorgio C Buttazzo. Hard real-time computing systems: Predictable scheduling algorithms and applications. *Computers & Mathematics with Applications*, 2003.

[11] Mahil Carr and June Verner. Prototyping and Software Development Approaches. *Prototyping and Software Development Approaches*, 2004.

[12] Alexander Clark, Chris Fox, and Shalom Lappin. *The Handbook of Computational Linguistics and Natural Language Processing*. 2010.

[13] J W Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 2014.

[14] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised, 2007.

[15] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT Solver. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008.

[16] Vinay Devadas and Hakan Aydin. On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-based Real-time Embedded Applications. *IEEE Transactions on Computers*, 61(1):31–44, 2012.

[17] David Devlin and Barry OSullivan. Satisfiability as a classification problem. In *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*, 2008.

[18] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering - ICSE '99*, 1999.

[19] Hamid Reza Faragardi, Aboozar Rajabi, Reza Shojaee, and Thomas Nolte. Towards Energy-aware Resource Scheduling to Maximize Reliability in Cloud Computing Systems. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 1469–1479. IEEE, 2013.

[20] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Herbert Zojer, and Christian Panis. DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *Proceedings of the 2011 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2011*, 2011.

[21] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A Compositional Framework for End-to-end Path Delay Calculation of Automotive Systems under Different Path Semantics. In *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.

[22] Orlando Ferrante, Luca Benvenuti, Leonardo Mangeruca, Christos Sofronis, and Alberto Ferrari. Parallel NuSMV: A NuSMV extension for the verification of complex embedded systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012.

[23] P. Filipovikj, N. Mahmud, R. Marinescu, C. Seceleanu, O. Ljungkrantz, and H. Lönn. *Simulink to UPPAAL statistical model checker: Analyzing automotive industrial systems*, volume 9995 LNCS. 2016.

[24] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. Analyzing Industrial Simulink Models by Statistical Model Checking. Technical report, 3 2017.

[25] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. Simppaal - A Framework For Statistical Model Checking of Industrial Simulink Models. *Submitted to Software Engineering and Methodology (TOSEM) Journal*, 2018.

[26] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. pages 104–124. Springer, Berlin, Heidelberg, 2008.

[27] Norbert E Fuchs and Rolf Schwitter. Attempto Controlled English {(ACE)}. *CoRR*, cmp-lg/960, 1996.

[28] A.L. Goel. Software Reliability Models: Assumptions, Limitations, and Applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, 12 1985.

[29] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A model for technology transfer in practice. *IEEE Software*, 2006.

[30] Ashlie B. Hocking, M. Anthony Aiello, John C. Knight, and Nikos Aréchiga. Proving Critical Properties of Simulink Models. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, 2016.

[31] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering*. Springer London, London, 2011.

[32] ISO26262 ISO. 26262: Road vehicles-Functional safety. Technical report, ISO/TC 22/SC 32 Electrical and electronic components and general system aspects, 2011.

[33] Thomas L. Harman James B. Dabney. *Mastering Simulink*. Pearson, 2003.

[34] Nikolaos Kekatos, Marcelo Forets, and Goran Frehse. Constructing verification models of nonlinear Simulink systems via syntactic hybridization. In *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, 2018.

[35] J. Kennedy, R. Eberhart, Carlos A Coello Coello, Gregorio Toscano Pulido, Maximino Salazar Lechuga, J. Kennedy, R. Eberhart, Carlos A Coello Coello, Gregorio Toscano Pulido, Maximino Salazar Lechuga, and From Scholarpedia. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995.

[36] Hermann Kopetz. Real-time systems: Design principles for distributed embedded applications. *Computers & Mathematics with Applications*, 2003.

[37] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real World Automotive Benchmarks for Free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[38] Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, 3 2014.

[39] Axel Legay, BenoÃ®t Delahaye, and Saddek Bensalem. Statistical Model Checking: An Overview. pages 122–135. Springer, Berlin, Heidelberg, 2010.

[40] Florian Leitner and Stefan Leue. Simulink Design Verifier vs. SPIN a Comparative Case Study. *Proceedings of FMICS*, 2008.

[41] N Mahmud, C Seceleanu, and O Ljungkrantz. ReSA Tool: Structured requirements specification and SAT-based consistency-checking. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1737–1746, 9 2016.

[42] N Mahmud, C Seceleanu, and O Ljungkrantz. Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017.

[43] Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Reza Faragardi, Saad Mubeen, and Cristina Seceleanu. Power-aware Allocation of Fault-tolerant Multi-rate AUTOSAR Applications. In *25th Asia-Pacific Software Engineering Conference*, 12 2018.

[44] Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Reza Faragardi, Saad Mubeen, and Cristina Seceleanu. Power-aware Allocation of Fault-tolerant AUTOSARSystems with End-to-end Timing Constraints viaHybrid Particle Swarm Optimization. Technical report, 2019.

[45] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. ReSA: An ontology-based requirement specification language tailored to automotive systems. In *2015 10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015 - Proceedings*, pages 1–10, 2015.

[46] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76, 2009.

[47] Karthik Manamcheri, Sayan Mitra, Stanley Bak, and Marco Caccamo. A step towards verification and synthesis from simulink/stateflow models. In *Proceedings of the 14th international conference on Hybrid systems: computation and control - HSCC '11*, page 317, New York, New York, USA, 2011. ACM Press.

[48] MathWokrks. Simulink Design Verifier.

[49] B. Meenakshi, Abhishek Bhatnagar, and Sudeepa Roy. Tool for Translating Simulink Models into Input Language of a Model Checker. pages 606–620. Springer, Berlin, Heidelberg, 2006.

[50] Seyedali Mirjalili. Particle swarm optimisation. In *Studies in Computational Intelligence*. 2019.

[51] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for End-to-end Response-time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and A Case Study. *Computer Science and Information Systems*, 10(1):453–482, 2013.

[52] Andriy Myachykov, Christoph Scheepers, Simon Garrod, Dominic Thompson, and Olga Fedorova. Syntactic flexibility and competition in sentence production: The case of English and Russian. *Quarterly Journal of Experimental Psychology*, 2013.

[53] Johanna Nellen, Thomas Rambow, Md Tawhid Bin Waez, Erika Ábrahám, and Joost-Pieter Katoen. Formal Verification of Automotive Simulink Controller Models: Empirical Technical Challenges, Evaluation and Recommendations. pages 382–398. Springer, Cham, 7 2018.

[54] Gerard OâRegan. Concise guide to formal methods, 2017.

[55] Alan Rector and Jeremy Rogers. Ontological and Practical Issues in Using a Description Logic to Represent Medical Concept Systems: Experience from GALEN. In Pedro Barahona, FranÃ§ois Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors, *Reasoning Web: Second International Summer School 2006, Lisbon, Portugal, September 4-8, 2006, Tutorial Lectures*, pages 197–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[56] Salah Eddine Saidi, Sylvain Cotard, Khaled Chaaban, and Kevin Marteil. An ILP Approach for Mapping AUTOSAR Runnables on Multi-core Architectures. In *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation Methods and Tools - RAPIDO '15*, pages 1–8, New York, USA, 2015. ACM Press.

[57] R Schwitter. English as a formal specification language. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, pages 228–232. IEEE Comput. Soc, 2002.

[58] Lui Sha, Tarek Abdelzaher, Karl Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective, 2004.

[59] Ivan Švogor, Ivica Crnkovic, and Neven Vrcek. An Extended Model for Multi-Criteria Software Component Allocation on a Heterogeneous Embedded Platform. *Journal of computing and information technology*, 21(4):211–222, 2014.

[60] Jiacun Wang. *Real-Time Embedded Systems*. 2017.

[61] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and SÃ©bastien Gerard. An Optimization Approach for the Synthesis of AUTOSAR Architectures. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2013.