Safety-critical systems need to be analyzed rigorously to remove software specifications errors, that is, their requirements specifications should be unambiguous, comprehensible and consistent, and the software design should conform to the specifications, hence avoiding undesirable system failures. Currently, there is a lack of effective and scalable methods to specify and analyze requirements, and formally analyze the behavioral models of embedded systems. Most embedded systems requirements are expressed in natural language, which is flexible and intuitive but frequently ambiguous and incomprehensible. Besides natural language, template-based requirements specification methods are used frequently to specify requirements (esp. in safety-critical applications). Although the latter reduce ambiguity and improve the comprehensibility of the specifications, they are usually rigid due to the constrained syntax of the templates, and template selection is challenging. Industrial systems are frequently developed by using modeling and simulation environments such as Simulink, which is also used to generate code automatically for various hardware platforms. Therefore, it is essential to be able to formally analyze Simulink models, to get insight into the behavior of the embedded system, and also prevent potential errors from propagating into the implementation. Analyzing the timing behavior of safety-critical software that is refined by multi-rate periodic tasks with data age constraints across the end-to-end software functionality is not trivial. This is due to the undersampling and oversampling effects caused by the data propagation from higher to lower rates and vice versa, respectively. Furthermore, when such systems are deployed on a distributed architecture, e.g., electrical/electronic vehicular system, besides assuring the timeliness, the reliability of the distributed software should be maximized to counter the higher risk of failures in the distributed computing setting, hence improving the overall predictability of the safety-critical system. However, designing for reliability usually requires additional critical system resources such as energy. Hence, to accommodate the growing complexity of software functionality, the design of the safety-critical systems should consider the efficient use of critical system resources such as the power source, while meeting the timing and reliability requirements.

To address the above needs, in this thesis, we propose formal-methods-based approaches and optimization techniques to assure improved quality of requirements specifications and software designs, and to efficiently map software functionality to hardware. The contributions of the thesis are: (i) ReSA - a domain-specific requirements specification language tailored to embedded systems, based on constrained natural language; (ii) a formal approach to check consistency of ReSA specifications via Boolean satisfiability problem (SAT) and ontology; (iii) a framework based on statistical model checking to analyze Simulink models via automated transformation into networks of stochastic timed automata; and (iv) a resource-efficient allocation of fault-tolerant software with end-to-end timing and reliability constraints via integer linear programming and hybrid particle-swarm optimization. Our proposed solutions are validated and 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 (EM) system benchmark from Bosch.

Nesredin Mahmud has been a PhD student at Mälardalen University, Sweden, since 2014. He conducts research at the Mälardalen Real-Time Research Centre (MRTC) in the area of embedded systems, formal methods and design space exploration. In particular, he applied various formal techniques and tools to address industrial challenges at Volvo Group Trucks Technology (VGTT), specifically to improve quality of automotive requirements specifications, and software designs modeled in Simulink. Furthermore, he has conducted research on the efficient deployment of distributed real-time software functionality on heterogeneous computing nodes by considering extra-functional properties such as timing, reliability and power consumption via integer linear programming and meta-heuristic algorithms.
DESIGN OF ASSURED AND EFFICIENT SAFETY-CRITICAL SYSTEMS

Nesredin Mahmud

2019

School of Innovation, Design and Engineering

Mälardalen University Press Dissertations
No. 291

MÄLARDALEN UNIVERSITY SWEDEN
DESIGN OF ASSURED AND EFFICIENT SAFETY-CRITICAL SYSTEMS

Nesredin Mahmud

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras torsdagen den 13 juni 2019, 13.15 i Gamma, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Joost-Pieter Katoen, RWTH Aachen University
Abstract

Safety-critical systems need to be analyzed rigorously to remove software/specifications errors, that is, their requirements specifications should be unambiguous, comprehensible and consistent, and the software design should conform to the specifications, hence avoiding undesirable system failures. Currently, there is a lack of effective and scalable methods to specify and analyze requirements, and formally analyze the behavioral models of embedded systems. Most embedded systems requirements are expressed in natural language, which is flexible and intuitive but frequently ambiguous and incomprehensible. Besides natural language, template-based requirements specification methods are used frequently to specify requirements (esp. in safety-critical applications). Although the latter reduce ambiguity and improve the comprehensibility of the specifications, they are usually rigid due to the constrained syntax of the templates, and template selection is challenging. Industrial systems are frequently developed by using modeling and simulation environments such as Simulink, which is also used to generate code automatically for various hardware platforms. Therefore, it is essential to be able to formally analyze Simulink models, to get insight into the behavior of the embedded system, and also prevent potential errors from propagating into the implementation. Analyzing the timing behavior of safety-critical software that is refined by multi-rate periodic tasks with data age constraints across the end-to-end software functionality is not trivial. This is due to the undersampling and oversampling effects caused by the data propagation from higher to lower rates and vice versa, respectively. Furthermore, when such systems are deployed on a distributed architecture, e.g., electrical/electronic vehicular system, besides assuring the timeliness, the reliability of the distributed software should be maximized to counter the higher risk of failures in the distributed computing setting, hence improving the overall predictability of the safety-critical system. However, designing for reliability usually requires additional critical system resources such as energy. Hence, to accommodate the growing complexity of software functionality, the design of the safety-critical systems should consider the efficient use of critical system resources such as the power source, while meeting the timing and reliability requirements.

To address the above needs, in this thesis, we propose formal-methods-based approaches and optimization techniques to assure improved quality of requirements specifications and software designs, and to efficiently map software functionality to hardware. The contributions of the thesis are: (i) ReSA - a domain-specific requirements specification language tailored to embedded systems, based on constrained natural language; (ii) a formal approach to check consistency of ReSA specifications via Boolean satisfiability problem (SAT) and ontology; (iii) a framework based on statistical model checking to analyze Simulink models via automated transformation into networks of stochastic timed automata; and (iv) a resource-efficient allocation of fault-tolerant software with end-to-end timing and reliability constraints via integer linear programming and hybrid particle-swarm optimization. Our proposed solutions are validated and 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 (EM) system benchmark from Bosch.
Abstract

Safety-critical systems need to be analyzed rigorously to remove software (or specifications) errors, that is, their requirements specifications should be unambiguous, comprehensible and consistent, and the software design should conform to the specifications, hence avoiding undesirable system failures. Currently, there is a lack of effective and scalable methods to specify and analyze requirements, and formally analyze the behavioral models of embedded systems. Most embedded systems requirements are expressed in natural language, which is flexible and intuitive but frequently ambiguous and incomprehensible. Besides natural language, template-based requirements specification methods are used frequently to specify requirements (esp. in safety-critical applications). Although, the latter reduce ambiguity and improve the comprehensibility of the specifications, they are usually rigid due to the constrained syntax of the templates, and template selection is challenging. Industrial systems are frequently developed by using modeling and simulation environments such as Simulink, which is also used to generate code automatically for various hardware platforms. Therefore, it is essential to be able to formally analyze Simulink models, to get insight into the behavior of the embedded system, and also prevent potential errors from propagating into the implementation. Analyzing the timing behavior of safety-critical software that is refined by multi-rate periodic tasks with data age constraints across the end-to-end software functionality is not trivial. This is due to the undersampling and oversampling effects caused by the data propagation from higher to lower rates and vice versa, respectively. Furthermore, when such systems are deployed on a distributed architecture, e.g., electrical/electronic vehicular system, besides assuring the timeliness, the reliability of the distributed software should be maximized to counter the higher risk of failures in the distributed computing setting, hence improving the overall predictability of the safety-critical system. However, designing for reliability usually requires additional critical system resources such as power and energy. Hence, to accommodate the growing complexity of software functionality, the design of the safety-critical systems should consider the efficient use of critical system resources such as the power source, while meeting the timing and reliability requirements.

To address the above needs, in this thesis, we propose formal-methods-based approaches and optimization techniques to assure improved quality of requirements specifications and software designs, and to efficiently map soft-
ware functionality to hardware. The contributions of the thesis are: (i) ReSA - a domain-specific requirements specification language tailored to embedded systems, based on constrained natural language; (ii) a formal approach to check consistency of ReSA specifications via Boolean satisfiability problem (SAT) and ontology; (iii) a framework based on statistical model checking to analyze Simulink models via automated transformation into networks of stochastic timed automata; and (iv) a resource-efficient allocation of fault-tolerant software with end-to-end timing and reliability constraints via integer linear programming and hybrid particle-swarm optimization. Our proposed solutions are validated and 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 (EM) system benchmark from Bosch.
Sammanfattning


För att möta ovanstående behov, föreslår vi i denna avhandling formella metoder och optimeringstekniker för att säkerställa förbättrad kvalitet pää kravspecifikationer och mjukvaruutveckling, och för att effektivt mappa mjukvarufunktionalitet till härdvara. Avhandlingens bidrag är: (i) ReSA - ett
domänspecifikt språk för kravspecifikation, skräddarsytt för inbyggda system, baserat på begränsat naturligt språk; (ii) ett formellt tillvägagångssätt för att kontrollera konsistensen av ReSA-specifikationer genom SAT och Ontologi; (iii) ett ramverk baserat på statistisk modellkontroll för att analysera Simulink-modeller via automatiserad omvandling till nätverk av stokastiska tidsautomater; och (iv) en resurseffektiv fördelning av feltolerant programvara med end-to-end-tidskrav och driftsäkerhetsbegränsningar genom heltals-linjär programmering och hybrid partikel-svärmoptimering. Väara föreslagna lösningar utvärderas i fall som används i fordon, såsom justerbar hastighetsbegränsare (ASL) och BBW-system från Volvo Group Trucks Technology (VGTT), och på ett motorstyrsystem från Bosch.
To My Father, Haji M. Jemal Mahmud
For Prioritizing Education and Pursuit of Learning!
Acknowledgements

I would like to express my deepest gratitude to my main supervisor Associate Professor Cristina Seceleanu. Your passion for research is inspiring, and your guidance, encouragements, determination, and patience have been the driving force of my PhD journey. It has been a pleasure to be around with you and get the best mentor one could ever get. I would like to thank my co-supervisor Oscar Ljungkrantz for the lively reception at Volvo Group Trucks Technology (VGTT), Gothenburg, and the wonderful researchers he introduced to me. It is my pleasure also to thank my co-supervisor Guillermo Rodriguez-Navas, for the great team work, valuable feedback, and encouragements. I am grateful to have you all as my supervisors. Your effort has made this work worthwhile.

Moreover, I would like to thank my examining committee Professor Peter Csaba Ölveczky, University of Oslo, Professor Wolfgang Ahrendt, Chalmers University of Technology, Associate Professor Raffaela Mirandola, Politecnico di Milano, for accepting our call and reviewing my thesis; and Associate Professor Antonio Cicchetti for reviewing the thesis proposal. Special gratitude goes to the faculty examiner Professor Joost-Pieter Katoen, RWTH Aachen University, for accepting the opponentship. It is an honour to have you all at my defense. Also, I would like to extend my appreciation to Damir Isovíc, Hans Hansson, Thomas Nolte, Moris Behnam, Kristian Sandström and the IDT administration staff, who facilitated my PhD defense and influenced my academic career.

I wish to acknowledge with much appreciation the role of Henrik Lönn, Oscar Ljungkrantz, as well as Saad Mubeen and Hamid Reza Faragardi, who supported my research technically at different levels, including co-authoring joint papers. Your willingness to share and support is highly admired. For the list is too long, I briefly express my respect and love to my colleagues at MDH, and outside such as Tewodros, Abu Hussaine, Björn. Your friendly support, jokes, and cultural exchanges will always be memorable.

I am thankful to Allah for granting me the well-being and strength to do my studies. I thank my parents for giving me unconditional love and supporting throughout my life, and my brothers, sisters and lifelong friends for the good words and encouragements. Thank you Big Brother Abdulwhab for being the example. In times of emotional stress, the support of my dear wife, Elham Salih, has been unconditional and unwavering. You are strong, confident and intelligent person and love of my life, and I thank the OOG for that! Our little boy, Mohammed, you should know that we love you so much. So learn the world with open mind and make us proud!
I would like to thank VINNOVA for sponsoring this research via the VeriSpec project, in which the financial support has made this work possible.

Nesredin Mahmud
Västerås, June 13, 2019
List of Papers Included in the Thesis

This thesis is based on the following papers:


Note: The research is submitted as a journal paper to Elsevier Journal of Systems Architecture (JSA). Under Review.

List of Papers Not Included in the Thesis


Reprints were made with permission from the publishers.
Contents

Part I: Thesis
1 Introduction ................................................................. 5
   1.1 Thesis Outline ....................................................... 8
2 Background ................................................................. 9
   2.1 Requirements Boilerplates ......................................... 9
   2.2 Logic-based Reasoning ............................................. 9
   2.3 Simulink ............................................................. 11
   2.4 Stochastic Timed Automata ....................................... 12
   2.5 Statistical ModelChecking ........................................ 14
   2.6 Integer linear Programming ...................................... 15
   2.7 Population-based Metaheuristics ............................... 16
3 Problem Formulation .................................................... 19
   3.1 Research Goals .................................................... 19
4 Research Method ......................................................... 23
5 Thesis Contributions .................................................... 25
   5.1 Design Workflow: Contributions Overview .................... 25
   5.2 RC1: ReSA – Requirements Specification Language ........ 27
   5.3 RC2: Consistency Checking of ReSA Specifications .......... 30
   5.4 RC3: Statistical Model Checking of Simulink Models ........ 32
   5.5 RC4: Fault-tolerant Software Allocation ...................... 34
   5.6 Included Papers and their Mapping to Contributions and Goals 41
6 Related Work ............................................................ 49
   6.1 Specification and Analysis of Embedded Systems Requirements 49
   6.2 Formal Analysis of Simulink Models ............................ 50
   6.3 Software Allocation Optimization ............................... 51
7 Conclusions and Future Work ........................................ 53
8 Bibliography .............................................................. 55

Part II: Included Papers
9 ReSA: An Ontology-based Requirement Specification Language Tailored to Automotive Systems ................................. 64
   9.1 Introduction ........................................................... 65
   9.2 Requirements Specification Methods ............................ 66
   9.3 Use-case: Adjustable Speed Limiter (ASL) ..................... 67
   9.4 ReSA: A Requirements Specification Language ............... 70
   9.5 Applying ReSA to ASL Requirements ............................ 82
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6 Requirements Analysis</td>
<td>83</td>
</tr>
<tr>
<td>9.7 Related Work</td>
<td>85</td>
</tr>
<tr>
<td>9.8 Conclusions and Future work</td>
<td>86</td>
</tr>
<tr>
<td>9.9 References</td>
<td>87</td>
</tr>
<tr>
<td>10 ReSA tool: Structured requirements specification and SAT-based consistency-checking</td>
<td>90</td>
</tr>
<tr>
<td>10.1 Introduction</td>
<td>91</td>
</tr>
<tr>
<td>10.2 Preliminaries</td>
<td>92</td>
</tr>
<tr>
<td>10.3 The ReSA Toolchain</td>
<td>96</td>
</tr>
<tr>
<td>10.4 Automated Consistency checking</td>
<td>100</td>
</tr>
<tr>
<td>10.5 Industrial Use Case: Adjustable Speed Limiter</td>
<td>104</td>
</tr>
<tr>
<td>10.6 Related Work</td>
<td>109</td>
</tr>
<tr>
<td>10.7 Conclusion</td>
<td>112</td>
</tr>
<tr>
<td>10.8 References</td>
<td>113</td>
</tr>
<tr>
<td>11 Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic</td>
<td>116</td>
</tr>
<tr>
<td>11.1 Introduction</td>
<td>117</td>
</tr>
<tr>
<td>11.2 Preliminaries</td>
<td>118</td>
</tr>
<tr>
<td>11.3 Defining ReSA Semantics in DL</td>
<td>121</td>
</tr>
<tr>
<td>11.4 Semantic Equivalence in ReSA</td>
<td>126</td>
</tr>
<tr>
<td>11.5 Automated Analysis and Transformation to TCTL Properties</td>
<td>127</td>
</tr>
<tr>
<td>11.6 Use Case: Adjustable Speed Limiter (ASL)</td>
<td>130</td>
</tr>
<tr>
<td>11.7 Related Work</td>
<td>132</td>
</tr>
<tr>
<td>11.8 Conclusion</td>
<td>133</td>
</tr>
<tr>
<td>11.9 References</td>
<td>135</td>
</tr>
<tr>
<td>12 SIMPPAAL - A Framework For Statistical Model Checking of Industrial Simulink Models</td>
<td>137</td>
</tr>
<tr>
<td>12.1 Introduction and Motivation</td>
<td>138</td>
</tr>
<tr>
<td>12.2 Preliminaries</td>
<td>140</td>
</tr>
<tr>
<td>12.3 Simulink to UPPAAL SMC: Approach</td>
<td>144</td>
</tr>
<tr>
<td>12.4 SIMPPAAL Tool</td>
<td>153</td>
</tr>
<tr>
<td>12.5 Application on Industrial Use Cases</td>
<td>158</td>
</tr>
<tr>
<td>12.6 Discussion on the Approach</td>
<td>163</td>
</tr>
<tr>
<td>12.7 Related Work</td>
<td>164</td>
</tr>
<tr>
<td>12.8 Conclusions and Future Work</td>
<td>167</td>
</tr>
<tr>
<td>12.9 References</td>
<td>170</td>
</tr>
<tr>
<td>13 Power-aware Allocation of Fault-tolerant Multirate AUTOSAR Applications</td>
<td>174</td>
</tr>
<tr>
<td>13.1 Introduction</td>
<td>175</td>
</tr>
<tr>
<td>13.2 AUTOSAR</td>
<td>176</td>
</tr>
<tr>
<td>13.3 System Model</td>
<td>177</td>
</tr>
<tr>
<td>13.4 Modeling Extra-functional Properties</td>
<td>180</td>
</tr>
<tr>
<td>13.5 Software Allocation Problem</td>
<td>183</td>
</tr>
<tr>
<td>13.6 Evaluation</td>
<td>188</td>
</tr>
</tbody>
</table>
13.7 Related Work .................................................. 193
13.8 Conclusions and Future Work ................................. 194
13.9 References .................................................... 195

14 Optimized Allocation of Fault-tolerant Embedded Software with End-to-end Timing Constraints ........................................ 199
14.1 Introduction .................................................... 200
14.2 AUTOSAR ..................................................... 202
14.3 System Model .................................................. 203
14.4 Software-to-Hardware Allocation Problem .................... 212
14.5 Solution using Hybrid Particle Swarm Optimization (PSO) ... 218
14.6 Evaluation ...................................................... 224
14.7 Related Work .................................................. 233
14.8 Conclusions and Future Work ................................. 234
14.9 References .................................................... 236
List of Tables

5.1 Mapping between the included papers and the research contributions. ........................................... 47
5.2 Mapping between the research contributions and the research goals. ........................................... 47
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Brake pedal and global brake controller of the Brake-by-Wire model.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>The Brake-by-Wire Subsystem block that computes vehicle speed. The labels in red color denote the execution order of the blocks.</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>STA example.</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Research Process.</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>Thesis contributions workflow.</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>The ReSA graphical user interface (GUI).</td>
<td>28</td>
</tr>
<tr>
<td>5.3</td>
<td>Specifying ASL requirements in ReSA.</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>The ReSA specifications R1 - R4 encoded as Z3 format, and the unsat-core feedback from the Z3 solver, to localize the source of the inconsistency.</td>
<td>31</td>
</tr>
<tr>
<td>5.5</td>
<td>STA transformation patterns.</td>
<td>33</td>
</tr>
<tr>
<td>5.6</td>
<td>Analysis results snippet of the BBW Simulink model from UPPAAL SMC. For the complete results, we refer the reader to [39].</td>
<td>35</td>
</tr>
<tr>
<td>5.7</td>
<td>ILP optimization of different sizes of software applications based on the number of software components, cause-effect chains, computing nodes.</td>
<td>39</td>
</tr>
<tr>
<td>5.8</td>
<td>Power consumption of nodes for different instances of software allocation problems by different optimization algorithms.</td>
<td>40</td>
</tr>
<tr>
<td>5.9</td>
<td>Computation time of the software allocation for different instances of the software allocation problems by different optimization algorithms.</td>
<td>41</td>
</tr>
</tbody>
</table>
Part I:

Thesis

This part of the document discusses the summary of the thesis, including the research problem, goals, and scientific contributions, followed by the research methods used in the thesis, related work, and conclusions and future work.
1. Introduction

REAL-TIME systems are characterized by timely computations, which are bounded by deadlines, besides correct results of the computations [23]. Many embedded systems are in fact real-time systems, especially due to their safety-critical nature. Safety-critical embedded systems are specialized computer systems designed for safety-critical applications [94], e.g. x-by-wire, flight control, industrial control systems. For instance, the brake-by-wire system [75] computes a torque proportional to the brake pedal force in order to slow down (or halt) the vehicle. Besides computing the correct torque, the system must act timely, that is not too soon and not too late so that traffic accidents are avoided. Therefore, safety-critical real-time systems should be analyzed for functional as well as timing correctness. In fact, they should be analyzed rigorously according to functional safety standards, such as ISO 26262 “Road vehicles-Functional safety” [48] that recommends the use of formal methods, which are mathematical techniques and tools that enable unambiguous specification, modeling and analysis of software and hardware systems [79][6], for instance, via model checking [9], theorem proving [51][3], rewriting logic [78], satisfiability modulo theories (SMT) [13].

Over the last decades, a lot of functionality that has been earlier realized by mechanical (or hydraulic) systems has moved to software, e.g. via the x-by-wire technology [75]. Moreover, several application software programs (or user-level software functions), which are developed due to innovation and compliance to safety standards, are integrated into safety-critical systems to provide advanced, safe and reliable services, for instance, the advanced driver-assistance systems (ADAS) [99][41], machine vision system for autonomous and self-driving vehicles [91][82].

Given this status quo, it is beneficial to ensure the quality of most artifacts produced during the development of safety-critical embedded systems, especially when such development is based on the model-based paradigm [77], which is the case for increasingly many industries. Consequently, in this thesis, we propose a design approach that considers rigorous analysis and efficiency of safety-critical embedded systems by applying formal methods and optimization techniques on various artifacts produced during at the early stages of software development.

One artifact to start with is the requirements specification [2] that entails documenting the functional and extra-functional (e.g. timing, reliability) requirements of a system in a comprehensive way. In practice, natural language
is the de facto standard for requirements specification, and quite seldom notations such as use cases, scenarios, UML diagrams, state machines. Even if natural language is intuitive and easy to use, requirements expressed in natural language can sometimes be ambiguous, vague, inconsistent, while intuitive to use [2]. In order to mitigate the aforementioned problems, several approaches that involve semi-formal and formal specification methods have been investigated. Template-based approaches [36], such as requirements boilerplates [47][36] and property specification patterns (SPS) [46][52], are semi-formal methods that use frequently reoccurring patterns, whereas other semi-formal methods rely on requirements meta-model [97][8][40][87]. The benefits of the template-based approaches are syntactic and semantic similarities to natural language with reduced ambiguity and a high degree of reuse. However, the methods are not scalable and template selection requires a lot of effort. In contrast, formal specification methods, e.g. temporal logics [44], Z language [80], etc., enable rigorous analysis but demand mathematical knowledge, which makes them not appealing to the average software engineers.

Due to the above, as a first contribution, we propose a constrained, yet flexible and unambiguous, natural language called ReSA, which is domain-specific and uses the notion of boilerplates to facilitate reuse [64]. The specifications created with ReSA have semantics in Boolean logic and description logic [69] to enable rigorous analysis via Boolean satisfiability [65] and ontology [19] methods, respectively. Therefore, as a second contribution, we propose a formal approach for analyzing ReSA specifications for logical consistency, via the two methods, which is implemented in our ReSA tool [60].

To complement the requirements modeling of an industrially-adopted architecture description language called EAST-ADL [18], the ReSA tool is integrated seamlessly into the Eclipse-based implementation of the EAST-ADL standard, called EATOP. The language and its tool support are validated on the adjustable speed limiter use case, which is a vehicle speed control system provided by Volvo Group Trucks Technology (VGTT) in Sweden.

The requirements specifications are used 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 [49] is one of the most widely used development environments for multi-domain, multi-rate, discrete and continuous safety-critical industrial systems. For this main reason, there is increasing interest in the formal analysis of Simulink models [66]. Simulink Design Verifier is the actual tool in the Simulink environment to formally verify Simulink design models. However, it supports only discrete models, and does not offer

---

1EAST-ADL - https://www.east-adl.info/index.html
2EATOP - http://synligare.eu/Tooling.html
support to verify timed properties, and high-level properties in general, such as reachability and invariance properties.

To address this lack, as a third contribution, we propose a scalable, formal approach for analyzing Simulink models, via statistical model checking, to verify functional and timing properties up to some probability. To automate the approach, we propose the tool SIMPPAAL [37][38] that uses UPPAAL Statistical Model Checker (UPPAAL SMC) [22] as the underlying model-checking engine. Our approach is pattern-based, and subsumes an execution-order-preserving automatic transformation of a Simulink model into a network of stochastic timed automata that can be formally analyzed by UPPAAL SMC [37]. The statistical model checker analyzes the generated state-transition system by conducting statistical analysis on the collected traces of the system executions [56], effectively avoiding the state-space explosion of (exact) model checking. To enable and ensure the correctness of the transformation, we first define the syntax and semantics of Simulink blocks and their composition, and prove the soundness of the transformation for a certain class of Simulink models. Our proposed technique is validated on the Brake-by-Wire use case, which is an industrial prototype provided by VGTT.

Due to the complexity of software that nowadays implements many system functions, several computing architectures have moved from federated architectures, that is, dedicated hardware systems executing safety-critical software systems or applications, into consolidated and distributed architectures, with multiple safety-critical software applications co-hosted on the same hardware. In distributed computing [53], the safety-critical software is mapped on multiple hardware systems to capitalize on the computational power provided by the distributed architecture, e.g. the Brake-by-Wire software executing on multiple electronic control units (ECU) that are connected through a network bus, such as CAN [31]. The distributed software, unlike federated computing, it is exposed to a greater degree of permanent and transient faults, therefore, the software reliability should be maximized to improve the overall dependability of the system. However, such measures require additional critical system resources besides computational resources, e.g. energy, which are constrained in battery-driven embedded systems. Thus, the distributed safety-critical software should be mapped onto hardware effectively, to satisfy the timing and reliability requirements of the software, but also efficiently to minimize the power consumption of the distributed system. We assume that the software is scheduled using a fixed-priority preemptive policy, which is quite common in industry, and possesses end-to-end timing requirements, e.g. the time duration between the brake-pedal press and the slowdown (or halt) of the vehicle is strictly bounded. Furthermore, we consider fault tolerance as a means to maximize reliability of the distributed safety-critical software by mapping redundant software functionality on different computing units.

As the fourth thesis contribution, we propose exact and heuristic optimization methods, which deliver optimal and near-optimal solutions, respec-
tively, to efficiently map the distributed safety-critical software to a network of computing nodes. Specifically, we propose a formulation of integer linear programming (ILP) problem [62], which is solved using the branch and bound method. Furthermore, we propose to solve the shortcomings of the exact method for large-scale problems [50], by using hybrid Particle Swarm Optimization (PSO) [72], which is a meta-heuristic algorithm, with trade-off over optimality. We show how to allocate AUTOSAR software applications on a network of heterogeneous computing nodes with respect to processor speed, failure rate and power consumption specifications, by employing both ILP and PSO. The ILP problem is implemented using ILOG CPLEX, which is a toolset for modeling and solving optimization problems. The proposed allocation approaches are validated on an automotive benchmark developed according to AUTOSAR, by Bosch, Germany.

1.1 Thesis Outline

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 on the logic-based reasoning using Boolean and description logic, Simulink, UPPAAL statistical model checking, and the optimization techniques based on integer linear programming and particle swarm optimization. In Chapter 3, we explain the research problem and outline the research goals. In Chapter 4, we describe the research method applied to conduct the research. The thesis contributions are discussed in Chapter 5, which also presents the mapping of the included papers to the contributions, and of the contributions to the research goals. In Section 6, we provide the related work on requirements specification, formal analysis of Simulink models, and software allocation. Finally, in Chapter 7, we conclude the thesis, outline the limitations and discuss possible directions for future work.

The second part of the thesis is a collection of papers that are included in this thesis and presented in Chapter 8 to Chapter 14.
2. Background

In this section, we give the necessary background on the different concepts and technologies used in this thesis, which includes requirements boilerplates, logic-based reasoning, Simulink, stochastic timed automata, statistical model checking, integer linear programming and population-based metaheuristics.

2.1 Requirements Boilerplates

Requirements boilerplates are frequently reoccurring patterns of statements to express software requirements. They have been first introduced to software engineering by Jeremy et al. [47] in order to improve requirements specifications by reducing ambiguities and enabling the reuse of specification patterns. A requirement boilerplate consists of static and dynamic syntactic parts. For instance, “if <Condition>, <System> shall be able to <Action> within <Quantity><Unit>.” is a conditional boilerplate. The syntactic elements annotated by pairs of angle brackets are dynamic (or variable), and the rest are static. The engineer has to select the right boilerplates from a repository that suits the target requirements, subsequently filling the dynamic syntactic elements to complete the specifications.

2.2 Logic-based Reasoning

In this thesis, we express requirements specifications in Boolean logic and apply Boolean satisfiability (SAT) to detect inconsistencies within the specifications. Furthermore, we use ontology to represent the specifications, and conduct rigorous analysis via a linguistic approach based on description logic. Consequently, below we briefly overview, SAT, and the notions of ontology and description logic.

Boolean Satisfiability Problem

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 com-
plete [34][17]. However, efficient heuristics and approximation algorithms do
exist to solve such problems. It is formally defined as follows:

**Definition 2.2.1** (SAT Problem). Given a propositional formula
\( \phi = (p_1, p_2, p_3, ..., p_n) \) over the logical connectives \( \neg, \lor, \land, \rightarrow \), where:
\( p_1, ..., p_n \) are atomic propositions, the SAT problem equates to checking if
there exists an interpretation (or a model) that satisfies \( \phi \), that is, a TRUE or
FALSE assignment to the variables of \( p_1, ..., p_n \), such that \( \phi \) evaluates to
true, in which case \( \phi \) is called satisfiable. In the opposite case, if no such
assignment exists, then \( \phi \) is unsatisfiable [34].

SAT problems are usually checked using decision procedures known
as SAT solvers. The Z3 tool integrates several background theories (or
Satisfiability Module Theories, SMT) and decision procedures based on SAT
solvers [32]. It is a well known SMT solver and a theorem prover developed
by Microsoft Research that has been used in the formal verification and
reasoning of software and hardware systems, mainly due to its strong support
for integration to verification tools via its well known APIs written, e.g. in
Java, Python, C.

The input language of Z3 is an extension of the SMT-LIB2 standard, con-
sisting of commands used to assert logical statements and instruct the solver
to do some action, e.g., to check for satisfiability, in which case the command
(*check-sat*) is invoked. Z3 returns *sat* if the formulas are satisfiable. If the for-
mulas are not satisfiable, Z3 returns *unsat*, or if it cannot decide the result, it
returns *unknown*. In case of *unsat*, if the unsat-core option is enabled, Z3 can
return the subset of unsatisfiable assertions by invoking the (*get-unsat-core*)
command.

**Ontology and Description Logic**

*Ontology* is a knowledge representation technique frequently used in the arti-
ficial intelligence and other domains to facilitate automated decision making.
It is defined as “a formal specification of a shared conceptualization” accord-
ing to Borst [20]. It employs different modeling entities, e.g., concepts, in-
stances, axioms, which are compared to classes, objects, inheritance in Object-
Oriented Programming. The ontology is consistent if all axioms and relations
(or assertions) hold [68].

*Description logic* (DL) [7] is a knowledge presentation language, and is
the most widely-used language to construct ontologies. It is a fragment of
first-order logic with many of the core reasoning problems decidable. For this
reason, it has several applications, e.g., in web semantics (e.g., OWL) [90],
artificial intelligence [16], bioinformatics [83], software engineering, natural
language processing. Hence, it is usually backed by solid reasoning services that terminate and deliver results in reasonable time.

2.3 Simulink

Simulink [49] 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 (that is continuous behavior) and global brake functionality (that is, discrete behavior), which execute every 10ms and 20ms.

![Figure 2.1: Brake pedal and global brake controller of the Brake-by-Wire model.](image)

Simulink is frequently used in the development of safety-critical systems, from the automotive, avionics or industrial automation systems domains, being often used to generate code automatically from the models. Therefore, it is crucial that such models are analyzed rigorously. The Simulink Design Verifier (SDV) [70] provides a formal verification technique that is based on the exact model checking to exhaustively verify low-level properties such as buffer overflow, division by zero etc. However, the tool’s functionality is limited as it lacks support for the analysis of high-level properties, such as invariance, reachability, timing, 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 functionality and consist of Input and Output ports, which enable communication via connectors. The modeling elements support basic and user-defined datatypes, e.g., integer, floating-point, and simple and complex data structures, namely 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 ex-
ecuted, hence do not affect the execution semantics of the model. The blocks can be composed into groups 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, like 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 systems, the framework supports mechanisms to extend functionality, which engineers can exploit to develop complex systems. The 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**

During the initial phase of the simulation in the Simulink environment, the model is compiled, thus the order in which the blocks are executed is established in the sorted order list (slist). 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. Basically, the list is determined according to the data dependency of block outputs on the block inputs ports, that is, if the output depends on the current value of the input, the input port is identified as direct-feedthrough port. Thus, to preserve the data dependency in the model, the sort-order rules require that the blocks that derive other blocks with 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., the Delay block in Figure 2.2, can execute in any order, considering that the previous rule applies. The execution order is also affected by the user-defined priorities, nevertheless, the priorities do not violate the rules. The sorted order list can be fetched by simulating the model in the debugging mode.

### 2.4 Stochastic Timed Automata

The theory of stochastic timed automata [28] builds upon the timed automata theory and adds probabilistic distributions to support stochastic behavior in modeling and analysis of real-time systems.

A timed automaton is a finite-state automaton proposed by Alur et al. [5] to model and analyze real-time systems, by extending finite-state automata with

---

Figure 2.2: The Brake-by-Wire Subsystem block that computes vehicle speed. The labels in red color denote of the execution order of the blocks.

Figure 2.3: STA example.

real-valued clock variables. A timed automaton (TA) \( \mathcal{A} \) is defined as a tuple \( L, l_0, X, \Sigma, E, I \), and its elements are described as follows: \( L \) is a set of locations of which \( l_0 \) is the initial location, \( X \) is the set of clocks, \( E \) is the set of edges between locations, \( \Sigma \) is a set of action labels, and \( I \) is a set of invariants, which are Boolean constraints over \( X \) assigned to locations, that is, the automaton can only stay in the location as long as its invariant holds. An edge \((l, g, a, r, l') \in E \) denotes a transition relation from location \( l \) to \( l' \), with guard \( g \), action \( a \) and clock resets \( r \). The guard is conjunction of clock constraints, and when it holds, the underlying transition corresponding to traversing the edge is fired.

A stochastic timed automaton (STA), as used in UPPAAL Statistical Model Checker (UPPAAL SMC) is defined by the tuple: \( STA = \langle \mathcal{A}, \mu, \gamma \rangle \), where \( \mathcal{A} \) is a timed automaton, \( \mu \) is the set of all density delay functions, given by either a uniform (for bounded delays via location invariants) or an exponential distribution (for unbounded delays), and \( \gamma \) is the set of all output probability functions over the output edges of the automaton.

Assume that, \( d(l, v) \) is the infimum delay that satisfies the disjunction of guards, and assume that, \( D(l, v) \) is the supremum delay before enabling and output. If the delay is bounded, i.e., there exists \( D(l, v) \), in location \( l \), each delay density function in \( l \), \( \mu_s \), is assumed to be uniformly distributed in the interval \( [d(l, v), D(l, v)] \). However, if the location is time unbounded, the delays are assumed to be exponentially distributed with the rate \( R(l) \). For more details on STA, we refer the reader to the literature [29].
Example 1 (STA Example). Figure 2.3 is an artificial STA example with three locations A, B and C, where A is the initial location. The guards of the automaton are colored in green (e.g., $x \geq 10$ over the edge that connects A and B), the clock resets are colored in blue (e.g., $x = 2$ on the same edge), and the location invariants are colored in purple (e.g., $x \leq 10$ on location C). The number in red on location A is the rate of exponential. The rate of exponential is in fact a user-defined value for the distribution parameter $\lambda$ in the delay function that calculates the probability that the automaton leaves the specified location at each simulation step, as follows: $Pr(\text{leaving location A after } t) = 1 - e^{-\lambda t} = 1000$. Basically, the greater the value of $\lambda$, the higher the probability that the automaton leaves the location.

Network of Stochastic Timed Automata

Under the assumption of input-enabledness, disjointness of clock sets and output action sets, a network of STA (denoted by NSTA) is parallel composition of $STA_i$ [29], that is $(STA_1 || STA_2 || \ldots || STA_n)$, where $n$ is the number of STA in the network. The state of an NSTA is a tuple $(s_1, s_2, \ldots, s_n)$, where $s_i = (l_i, v_i)$, where $l_i$ is a location of $STA_i$, and $v_i$ is the valuation of clocks $X_i$. In NSTA, the automata communicate through broadcast channels and globally shared variables. Moreover, the semantics of the NSTA relies on the independent computation of the individual STA (or components), i.e., each component automaton, based on the delay density functions and the output probability functions decides repeatedly on which output to generate and at which point in time. In this race, the output will be determined by the component automaton that has chosen to produce the output after the minimum delay [28].

2.5 Statistical Model Checking

Statistical model checking [4] uses a finite set of randomly selected execution traces (or runs) of the state-based system $S$ to check if the traces provide probability evidence for accepting or rejecting the specification (or property) $\varphi$, i.e., $T \in \text{Traces}(S) \models \varphi$. It applies statistical techniques such as hypothesis testing and Monte Carlo simulation to compute the probability evidence. Since exhaustive model checking encounters the state-space explosion problem for large models, statistical model checking is a viable alternative to exhaustive verification via probability estimation.

UPPAAL Statistical Model Checker (UPPAAL SMC) [22] is an extension of the model checker UPPAAL with support for statistical model checking. It admits a system modeled as NSTA, and the properties to checked are specified in probabilistic weighted metric temporal logic (PWMTL) [28], which is the probabilistic extension of weighted metric temporal logic (WMTL). The
The properties are of the following form:
\[
\phi ::= P(\diamond x \leq T \phi) \triangleright p|P(\Box x \leq T) \triangleright p \tag{2.1}
\]
where \(x\) is a clock, \(T \in \mathbb{N}\) is a bound, \(\phi\) is a state predicate, \(\triangleright \in \{<, \leq, =, >, \geq\}\), and \(p \in [0, 1]\).

UPPAAL SMC [30][22] can be used to check qualitative properties such as hypothesis testing and probability comparison, as well as quantitative ones via probability estimation. The property \(\phi\) of Equation (2.1) shows the definition of hypothesis testing. The probability of making hypothesis testing errors is bounded by the strength parameters \(\alpha\) and \(\beta\), where \(\alpha\) is the probability of accepting the null hypothesis while its opposite holds (false negative), and \(\beta\) is the probability of accepting the opposite of the null hypothesis while the null hypothesis holds (false positive) and \(p\) is the estimated probability of the property holding. Another qualitative property that can be checked using UPPAAL SMC is the comparison of probabilities.

The probability evaluation is a quantitative method that determines the probability of satisfying a PWMTL property \(\phi\) of an STA. The evaluation is an approximation interval \([p - \mu, p + \mu]\) with a confidence \(1 - \alpha\), where \(\mu\) and \(\alpha\) are user configurable parameters. The number of traces is determined automatically by the model checker, based on the parameters.

### 2.6 Integer linear Programming

An integer linear programming (ILP) [21] is a type of optimization problem that consists of integer decision variables, and the linear objective function and constraints. The binary ILP problem is a special case of ILP where decision variables are binary. It is represented as:

\[
\text{Maximize } \sum_{j=1}^{n} c_j x_j \tag{2.2}
\]

Subject to:
\[
\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for all } i = 1, \ldots, m \tag{2.3}
\]
\[
x_j \in \{0, 1\} \quad \text{for all } j = 1, \ldots, n \tag{2.4}
\]

where (2.2) is the objective function with \(n\) cost coefficients \(c_j : j = 1, \ldots, n\), (2.3) are \(m\) linear constraints with coefficients \(a_j : 1, \ldots, m\) and bounds \(b_j : 1, \ldots, m\), and (2.4) are binary decision variables with \(x_j : 1, \ldots, n\).

ILP is used in many control and engineering applications, e.g., fleet and flight routing in transportation, airfoil design in avionics engineering, controller design in industrial automation, automotive, etc., resource allocation.
The binary variant is applied in several resource allocation problems, like assignment of operating system tasks to processors. The software allocation, for instance, in component-based development, can be formulated as a binary integer programming problem, where $x_j = 1$ denotes the fact that a component is mapped to the $j^{th}$ processor and $x_j = 0$ denotes the fact that a component is not mapped to the $j^{th}$ processor.

ILP problems are frequently solved via exact algorithms, e.g., branch and bound, but can also be dealt with heuristics, e.g., using simulated annealing, hill climbing. In this work, we use the CPLEX solver \(^2\) form IBM to the software allocation optimization.

### 2.7 Population-based Metaheuristics

The ILP approach and for that matter exact methods do not scale well on complex optimization problems, or else their application is usually prohibitively expensive when applied on large-scale problems, i.e., with many decision variables. In contrast, metaheuristics [1][43], which is a type of heuristics with search strategy, is more efficient computation-wise, albeit less optimal. The population-based metaheuristics is a class of meta-heuristic algorithms which uses a set of individuals (or population) to guide the search strategy and determine the global optima of the problem. The most common population-based algorithms consist of evolutionary algorithms, differential evolution [92][27], particle swarm optimization [81][72].

In this thesis work, for the efficient software allocation, we apply the differential evolution, particle swarm optimization (PSO) and the latter’s hybrid with local search algorithms such as hill climbing and stochastic hill climbing to find the best (or near optimal) solution. In the case of differential evolution, special evolutionary operators, such as mutation, crossover and selection, are employed to update the population, thus traverse across the search space to reach at the global near-optima. The technique is briefly described as follows. A mutant agent (or individual), i.e., the offspring $v$, is generated for every agent from three other agents in each iteration according to (2.5), subsequently combined using the crossover operator (2.6). If the offspring $u$ is more fit than the agent $x$, i.e., closer to the best solution, it replaces the agent otherwise is discarded as shown in the selection strategy (2.7).

$$v \leftarrow a + F \circ (b - c) \quad (2.5)$$

$$u \leftarrow \text{crossOver}(v, x, CF, F) \quad (2.6)$$

$$x \leftarrow \begin{cases} u & \text{if } f(u) < f(x) \\ x & \text{otherwise} \end{cases} \quad (2.7)$$

---

\(^2\)https://www.ibm.com/analytics/cplex-optimizer
where $F \in [0, 2]$ is known as differential weight and controls the diversity of the mutant, and $CF \in [0, 1]$ is known as the crossover probability and controls the degree of crossover between the agent and the mutant.

Similarly, the particle-swarm optimization employs individuals (or particles in this case). The particles unlike in the differential evolution are memory-based, i.e., they record their best position so far $p_{bst}$, and the algorithm remembers the best position of the population $z$. The motion of each particle is guided by its velocity (2.8) and attractions towards its best position $p_{bst} - p$ and towards the best position of the population $z - p$ (2.9), where $p$ is an $n$-dimensional matrix which represents the current position of a particle.

$$v \leftarrow \omega v + c_1 Rand() \circ (p_{bst} - p) + c_2 Rand() \circ (z - p) \quad (2.8)$$

$$p \leftarrow p + v, \quad (2.9)$$

where $\omega$ 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

Over the last decades, the complexity of the safety-critical software has been rising in most embedded systems. This is easily evident in modern cars, which implement many and complex automotive functions, as well as with the emergence of the electrical and autonomous vehicles. Thus, the thesis is motivated by the need for advanced (or rigorous) methods to specify, model and analyze complex safety-critical automotive software, and their seamless integration into industrial practice, at heavy-vehicle manufacturers such as VGTT and Scania. Furthermore, the thesis is motivated by the need for efficient mapping of safety-critical software to hardware in a distributed computing setting to facilitate software extensibility and support the increasing functionality of the automotive systems.

The overall goal of the thesis is as follows:

**Overall Goal** — Provide assurance of functionality and quality of embedded software systems, at various levels of abstraction, via formal analysis and optimization of critical system resources.

The overall goal is refined via research goals, which narrow the overall goal by splitting it into smaller and more concrete goals deemed essential to achieve the overall goal.

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 (requirements boilerplates [36][64]), controlled natural languages [55](e.g., Attempto [40]).

The template-based methods inherently lack meta-model (or grammar), therefore, it 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 defined as follows:
**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 [55]. 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 [2][74], 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 (like logical contradictions). However, natural language lacks formal (or precise and unambiguous) semantics, therefore is difficult to rigorously analyze natural-language requirements specifications. There are several methods to natural language semantics, for which the use of logic is common [25].

Therefore, we define the second research goal as follows:

**RG 2:** — Facilitate logic-based analysis of embedded systems requirements specifications.

Natural language specifications are constructed from syntactic units, such as words, phrases, clauses, statements, etc. Consequently, the rigorous analysis of specifications involve parsing and interpreting the syntactic units, which is a complex problem in computational linguistics [25]. The depth of the interpretation (or semantics) affects the applicability of the methods a great deal, for instance, 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 rigorous, thus enable thorough analysis, yet less tractable. Therefore, an 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 used for modeling and simulating the behavior of multi-domain, discrete, and continuous embedded systems. Simulink also enables the generation of code from discrete Simulink models which directly execute on specific platforms.

Consequently, it is crucial to conduct rigorous analysis of Simulink models to reduce the number of errors introduced in the generated code. The de facto Simulink analysis techniques by type checking, simulation, and formal veri-
fication via 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 it also lacks support for verifying continuous models and suffers from scalability due to its reliance on the exhaustive model-checking [58]. In contrast to exhaustive model checking, the statistical model-checking verifies properties over sufficiently many traces of system simulations collected via statistical methods. Such analysis approach makes the technique highly scalable, if compared to exhaustive model checking.

Our third research goal is formulated as follows:

**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 [49]. 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 continuous Simulink blocks, making the model checking challenging.

In distributed architectures, the automotive software is allocated on multiple computing units (or ECU), it is exposed to permanent and transient faults, hence the need to maximize the reliability of the safety-critical software system. Fault tolerance using redundancy is the most popular approach to improve reliability by replicating software functionality on multiple ECU. However, designing for fault tolerance requires additional critical system resources such as power sources. In this regard, the software-to-hardware allocation plays a crucial role in minimizing the power consumption of fault-tolerant distributed safety-critical software, without breaching the timing and reliability constraints of the safety-critical software.

Given the challenges, we formulate the fourth research goal as follows:

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

The software allocation model is not trivial as we consider an exact method of schedulability analysis and reliability calculations, which makes the optimization complex. We consider the worst-case response time analysis [14][31] to check schedulability of the tasks, and assume age constraints on the cause-effect chains [73], which is practical but computationally expensive. For the

---

$^1$https://se.mathworks.com/products/sldesignverifier.html
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.

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

Thus, the last research goal is:

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

The integration of our proposed methods and tools into the existing development process requires close cooperation between the domain experts and the practitioners. The role of the domain experts should be to simplify usage of the tools, for instance by rendering their interface to existing ones, 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 of being reserved in applying them. In this thesis, we also target the integration of some thesis results into in-house tools at VGTT.
4. Research Method

Research methods, according to Jane et al. [15], are approaches, procedures and guidelines that are applied to conduct research. Out of them, observation, interview, prototyping, experiment, are just a few. 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. 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 engineer-friendly, which means the research team is required to communicate with the actual users that capture the feel of the proposed methods and tools.

![Figure 4.1: Research Process.](image)

Figure 4.1 illustrates the research process employed in the thesis, which is an adaptation of the technology-transfer process proposed by Tony et al. [45]. Our main research activities have involved identification of research goals and research challenges, which are followed by solution proposals and their imple-
mentations. Subsequently, the solutions are validated on industrial use cases, and some 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 [26]. 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.

During implementations of the proposed solutions, we have applied a prototyping method [24], 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 [60, 61] as well as the SIMPPAAL framework [37]. 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.

1Work in progress - validation of the ReSA toolchain by practitioners, at VGTT.
5. Thesis Contributions

The overarching contribution of the thesis is a design approach targeting safety-critical embedded software, which employs formal methods at various stages of software development, such as requirements specification and software design, and a power-efficient allocation of software to distributed hardware. Our approach meets the research goals defined in Section 3, via the following contributions: a requirements specification language tailored to embedded systems [64], formal analysis of the specifications [60][61] via logic-based methods, formal analysis of large-scale Simulink models via statistical model checking [39], and efficient mapping of software to hardware in the context of distributed architectures, via integer linear programming and hybrid particle swarm optimization [62][63].

5.1 Design Workflow: Contributions Overview

In Figure 5.1, we illustrate the workflow of an embedded software development that makes use of our contributions. The workflow is contextualized in the automotive systems development where Simulink\(^1\), as well as architectural languages such as EAST-ADL\(^2\) and AUTOSAR\(^3\) are used.

Initially the textual requirements are specified in our domain-specific language, ReSA, which is a constrained natural language tailored to embedded systems. The ReSA editor supports content completion as it is connected to a system model, which contains instantiations of words/phrases. If the ReSA editor is required to be domain-specific, e.g., to the automotive domain that employs, EAST-ADL, the system model is specialized to the latter model, thus enabling requirements specifications by accessing EAST-ADL model elements from the editor. To check the consistency, the ReSA specifications are translated into Boolean and description logic, respectively, and subsequently analyzed via SAT solver and inference engine (or reasoner), respectively.

After the quality of the specifications is checked and improved if needed, that is made, unambiguous, comprehensible, consistent, they can be used in the verification of behavioral software designs. In this thesis, we consider that

\(^{1}\text{Matlab Simulink - https://se.mathworks.com/products/simulink.html?requestedDomain=}\
\(^{2}\text{EAST-ADL - https://www.east-adl.info/}\
\(^{3}\text{AUTOSAR - https://www.autosar.org/}\

the software design is created with Simulink, which is one of the most popular integrated environments for modeling, simulation and analysis of embedded system functions. In order to analyze a Simulink model rigorously, we transform it automatically into a formal model, specifically a network of stochastic timed automata. Subsequently, the latter is analyzed via statistical model checking against properties initially expressed in ReSA, but manually translated into the query (properties) language of the model checker (UPPAAL SMC), that is, probabilistic weighed-metric temporal logic (PWMTL).

At the architectural level, the Simulink model is represented by EAST-ADL, and at the implementation level by AUTOSAR, which consists of software components that communicate through a virtual function bus (VFB). Note that, the refinement from the AUTOSAR model to the Simulink model is manual, which is the normal practice, however, the Simulink model composition (or structure) is automatically generated. The software architecture complements the Simulink model with computation resource specifications, that is via runnables, which are schedulable pieces of code (or objects) by the AUTOSAR OS. Moreover, AUTOSAR also complements Simulink models with resource specifications, such as power specification, interface specification of the underlying hardware, etc. In this regard, we propose exact and heuristic software-to-hardware optimization techniques that incorporate the timing and reliability of software applications as constraints, and optimize the power consumption of the software.

The thesis contributions are summarized in the next sections.
5.2 RC1:ReSA – Requirements Specification Language

We propose a domain-specific and constrained natural language, both at the syntax and semantics level, which is designed to improve comprehensibility and reducing ambiguity of embedded systems requirement specifications. The language employs concepts from the embedded systems domain, e.g., System, Parameter, Device, State, Mode, Entity, to typeset domain-specific words/phrases. Besides grammatical rules (or syntax), semantic relations between the concepts limit the possible construction of the specifications. For instance, the sentence “The ASL shall limit the driver” is a grammatically-correct specification, but semantically it makes no sense. To solve this problem, the ReSA type system imposes a type constraint on the relation between instances “The ASL”:system and “the driver”:user.

Syntax

The language design is modular, in the sense that it uses sentence categories, e.g., Simple, Compound, to structure the specifications, thus improving comprehensibility and reduce ambiguity. It also allows complex specifications by inductively applying the grammar rules of the language. The following context-free grammar is a snippet of the ReSA language.

\[
\begin{align*}
\langle \text{specification} \rangle & \mid \langle \text{simple} \rangle | \langle \text{complex} \rangle | \langle \text{nested-complex} \rangle \\
\langle \text{simple} \rangle & \mid \langle \text{mainClause} \rangle. \\
\langle \text{complex} \rangle & \mid \langle \text{subClause} \rangle, \langle \text{mainClause} \rangle. \\
\langle \text{mainClause} \rangle & \mid \langle \text{primitiveClause} \rangle | \ldots \\
\langle \text{primitiveClause} \rangle & \mid \langle \text{sub} \rangle \langle \text{verb} \rangle \langle \text{obj} \rangle | \ldots \\
& \ldots
\end{align*}
\]

For the detailed grammar specification of the language, the reader is invited to refer to Paper A [64], included in this thesis, and the ReSA documentation. ⁴

Example 2 (Adjustable Speed Limiter (ASL) Requirements). We here give an example of selected requirements of the ASL system, which is a speed control system found in Volvo trucks. It controls the vehicle speed to not exceed a certain speed, which is set by the driver or legal authorities. It consists around 300 functional and extra-functional requirements. To demonstrate the use of ReSA, we specify only four of ASL requirements.

⁴ReSA documentation: https://bitbucket.org/nasmdh/resa/src/master/
R1 ASL:system shall send "the driver":user notification:status every 200ms.
R2 if “driver:user” 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 dis-enabled.

Figure 5.2: The ReSA graphical user interface (GUI).

Tool Support and Validation on Adjustable Speed Limiter
The ReSA language is implemented [60] in the Xtext framework\(^5\), which is an Eclipse-based integrated development environment for the development of domain-specific and general-purpose languages. The ReSA editor, which is shown in Figure 5.2, supports content completion, errors/warring messages and boilerplate management. The editor is integrated in EATOP, meaning that the editor can be triggered from within the IDE, moreover, the content-completion feature displays contents of the EAST-ADL model elements, hence enabling the consistent use of vocabularies during specification. The ReSA

\(^5\)Xtext:https://www.eclipse.org/Xtext/
tool as a standalone, and as a plugin for EATOP, can be downloaded from the URL https://bitbucket.org/dashboard/overview.

The language as well as its tool support is validated on the 300 ASL requirements, which are initially expressed in natural language. The requirements distribution is as shown in Figure 5.3a, in terms of types of requirements. The requirements are rewritten in ReSA, and Figure 5.3b shows the sentence categories (or boilerplates) employed to specify the requirements. In general, the language is expressive, though it is constrained, and is extensible with constructs that improve its expressiveness.

(a) Types of ASL Requirements.

(b) Distribution of ASL boilerplate types.

Figure 5.3: Specifying ASL requirements in ReSA.
5.3 RC2: Consistency Checking of ReSA Specifications

The editor shows syntax and typing warning and error messages. However, such static analysis is rudimentary and is per specification. In this contribution, we propose two methods of consistency analysis: SAT-based analysis and ontology-based analysis, which analyze multiple specifications for lack of logical contradictions.

SAT-based Analysis

Boolean satisfiability (also known as SAT) can informally be defined as the problem of finding truth values (or assignments of Boolean variables) such that a Boolean formula holds. It is NP-complete, however, there exist efficient heuristic algorithms that solve SAT problems of large size, e.g. with thousands of Boolean variables. Therefore, we first transform the ReSA specifications into Boolean expressions [60], then we formulate the problem of finding inconsistencies of the Boolean expressions, in terms of SAT as follows:

Definition 5.3.1 (Inconsistency of Specifications). Let \( \phi = \{ \phi_1, \phi_2, \ldots, \phi_n \} \) be the ReSA specification, with its constituents translated into Boolean expressions, \( \phi_i, i \in [0, \ldots, n] \). Consequently, \( \phi_1, \phi_2, \ldots, \phi_n \) are Boolean formulas, each denoting a requirement respectively. We say that the set of Boolean formulas \( \phi \) is inconsistent if the implication \( \phi_1 \land \phi_2 \land \cdots \land \phi_n \Rightarrow false \) holds, that is, there exists at least one expression that cannot be satisfied: \( \exists i. \phi_i \mid = false \).

Example 3. Assume that we want to check the consistency of the requirements that are specified in Example 2. Figure 5.4 shows the translation of the specifications into Boolean expressions (in Z3 SMT-LIB format). Each expression is asserted before the Z3 solver is called using the command (check - sat). Some clauses are resolved for negation and opposite words, e.g., the fact that “enabled” is opposite of “disabled” implies \( p_5 = p_7 \neq p_{11} \). The solver returned “unsat (R2 R4 R1_Assertion)”, which means, the expressions are inconsistent and indicates the part of the specification that is responsible for the inconsistency (or unsat-core).

The SAT-based analysis, although easy and scalable, does not allow rigorous analysis. This is due to the fact that the Boolean (or propositional) variables abstract away details of the clauses in the specifications.

Ontology-based Analysis

Ontology as a knowledge representation technique can be used to capture the intricate relations of words/phrases of specifications, for instance, synonyms, antonyms, generalization. Moreover, it can be used to capture the syntax of the specifications as well, essentially modeling the ReSA specification as a concrete requirements specification ontology.
Figure 5.4: The ReSA specifications R1 - R4 encoded as Z3 format, and the unsat-core feedback from the Z3 solver, to localize the source of the inconsistency.

Besides the challenge of constructing the ontology, accurate representation of the specifications is crucial to a certain degree for the ontology to be useful, that is, the specifications should be interpreted linguistically. In this research, we propose the event-based semantics approach [61] to construct the meaning of each specification compositionally from its constituents (or grammar units). 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 [61]. To help intuition, let us assume the following example: the clause “ASL:system shall limit "vehicle speed":parameter” is represented as $\exists e [\text{limiting}(e) \& \text{Agent}(ASL) \& \text{Recipient}(vehicle speed)]$, where Agent and Recipient are known as thematic roles, which define the semantic roles of the arguments in the clause. Since the requirement specifications entail technical words/phrase, the thematic roles of these arguments are not obvious, so we extended the thematic roles into the ReSA concepts, in this way, the interpretation of the clauses becomes domain-specific, for example, classifying instances of System or User as Agent, and instances of Parameter as Recipient.

Following the event-based semantics, the ReSA specifications are translated into description logic as ontology. Once the ontology is constructed considering the event-based semantics, thematic roles, lexical relation, we check its consistency 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 5.3.2 (Inconsistency of Specifications). The requirements specification ontology is inconsistent if there does not exist an interpretation (or a
model) $M$ that satisfies the terminological assertions $T$ and the concrete assertions $A$, that is, $M \nvDash ax$, where: $ax \in T \cup A$, where $T$ is a set of terminological assertions and $A$ a set of facts (or concrete assertions).

5.4 RC3: Statistical Model Checking of Simulink Models

Several research endeavors do exist on the topic of formally analyzing Simulink models. Some of them are using theorem proving, others exact model checking and others statistical model checking. Many of the proposed solutions are limited either due to frequent user involvement or scalability issues. In this thesis, we propose a scalable formal approach for analyzing large-scale Simulink models via statistical model checking. The Simulink models can be discrete, continuous or hybrid, moreover, they can consist of atomic and composite Simulink blocks that are scattered on multiple files, which is a practical scenario. Essentially, we consider typical industrial Simulink models, e.g., the Brake-by-Wire and Adjustable Speed Limiter Simulink models from VGTT [39].

To enable statistical model checking of Simulink models, we first define the syntax and semantics of blocks and their composition, in terms of timed transition systems. After that, we transform the Simulink blocks of a model, after flattening, in networks of stochastic timed automata that can be model checked statistically with UPPAAL SMC. Our approach has the following important characteristics: (i) the transformation employs continuous and discrete transformation patterns, which are generic and reusable, thus can be applied to any Simulink blocks; (ii) the transformation preserves the execution order of Simulink blocks; (iii) it is robust, meaning that, it handles various models such as continuous, discrete, hybrid, and models that contain blocks with with different sampling rates, also known as multi-rate blocks.

Transformation Patterns

The continuous-time and discrete-time stochastic timed automata (STA), which are shown in Figures 5.5(a) and 5.5(b), are used to transform the continuous-time and discrete-time Simulink blocks into their respective STA. In both cases, the automata traverse the respective edges from the “Start” locations to the “Operate” locations, respectively, at the global time $sn \ast IAT$, which is determined according to each block’s order of execution $sn$, that is, the lower $sn$, the sooner the automaton moves to the “Operate” location, $IAT$ is an infinitesimal inter-arrival time between subsequent executions of the automata. In case of continuous-time STA pattern, the automaton loops at the “Operate” location every infinitesimal sample time, which is distributed exponentially according to the rate $\lambda = 1000$. In the discrete-time STA pattern, the automaton loops at location "Operate" every $ts$ sample time with
probability 1, as there is only one edge that goes out of the location and the delay transitions are distributed uniformly in the interval \([t_s, t_s]\).

![Figure 5.5: STA transformation patterns.](image)

(a) Continuous-time. (b) Discrete-time.

Simulink to NSTA Transformation Process

Besides proposing the transformation patterns, we provide an automatic transformation of a Simulink model to its equivalent network of STA by applying the following tasks: (i) the Simulink model is simulated, subsequently, the sorted-order list, which contains the execution order of the Simulink blocks, is extracted; (ii) in case the model is hierarchical, the execution order is flattened first, thus generating a flattened sorted-order list; (iii) the Simulink model is parsed, as a result, each block is translated into a stochastic timed automaton via the corresponding transformation pattern. The sorted-order number \(sn\) is instantiated from the sorted-order list on the fly; (iv) any non-computational block is accounted in the transformation but is not transformed into an automaton; examples of such blocks are Mux, SubSystem blocks etc.; (v) the connections between the blocks are translated into global variables, which act as the communication means between the automata. For details on the transformation, as well as on the soundness proof for discrete-time models, the reader is invited to consult Paper D [39], also included in this thesis.

Validation on the Brake-by-Wire Use Case

Our approach is applied on an industrial Brake-by-Wire Simulink model, which is a prototype developed for academic use by VGTT. The model consists of 320 blocks, of which 19 are discrete blocks, 26 constant blocks and the rest are continuous blocks. The model design consists of modules that collect sensory data, e.g., vehicle speed, pedal position, and compute the proportional torque force that is applied on the wheels to brake the vehicle. Besides, it has the anti-braking functionality (ABS) to avoid (or minimize) skidding, which occurs when the slip rate of a wheel is greater than its friction coefficient.
The BBW requirements are initially specified in ReSA, subsequently are translated into PWMTL properties manually by using monitors (or observers), which are stopwatch stochastic priced timed automata. The monitors are modeled for particular requirements, and basically observe the progression of the execution of automata.

**Example 4** (Statistical model checking of BBW). The BBW model is automatically transformed into a network of STA, and we model monitors [39] to represent 5 functional and timing requirements, out of which 3 requirements are shown in the box below for illustration.

<table>
<thead>
<tr>
<th>R1_{BBW}</th>
<th>The brake request shall be computed within 10 ms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3_{BBW}</td>
<td>The brake request shall be propagated to two different wheel actuators within 4 ms.</td>
</tr>
<tr>
<td>R4_{BBW}</td>
<td>If the brake request is 0, then the ABS shall set the torque to 0.</td>
</tr>
</tbody>
</table>

Figure 5.6 shows the PWMTL properties and their model-checking results. R1_{BBW} and R3_{BBW} as presented in the able above are safety requirements expressed using the $\square$ (always) operator. The first R1_{BBW} is in the form of hypothesis testing that was verified on a pruned version of the BBW formal model that contained only the parts relevant to the property. The second R1_{BBW} is a probability estimation of a stronger version of the property in which we check the reachability of the End location of the Monitor automaton, and also the invariance property that states that the clock should never exceed value of 10. Since both are satisfied the actual R1_{BBW} is also satisfied. R4_{BBW} is a conditional requirement expressed using the $\hat{\diamond}$ (eventually) operator in its formal translation, as encoded by the R4_{BBW} PWMTL property. Each result in Figure 5.6 shows the probability interval of satisfying the property, confidence level, the number of runs and the time needed by the model checker to return the result. For some properties in Table 5.6, the probability interval of satisfying the respective property is $[0.99, 1]$, which can be considered as “property satisfied” if the lower bound on the probability is 0.99. However, in many safety-critical applications, the requirement on the probability is normally high, e.g., 0.999999 (essentially approximates to “no failure should happen”). In this case, the interval should be more precise, which can be achieved by lowering $\alpha$ and $\beta$ [28].

5.5 RC4: Fault-tolerant Software Allocation

As the fourth contribution of this thesis, we have proposed, we have proposed two software allocation approaches based on integer linear programming (ILP) and hybrid particle swarm optimization (PSO) optimization algorithms. The system model is briefly described as follows. We consider an AUTOSAR safety-critical software application that can be mapped to multiple...
<table>
<thead>
<tr>
<th>Req.</th>
<th>Query</th>
<th>Result</th>
<th>Runs</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( R_{1BBW} ) ( Pr<a href="Monitor.End">&lt;= 35</a> ) ( \implies Monitor.x &lt;= 10 ) ( &gt;= 0.999 )</td>
<td>( Pr([...]) &gt;= 0.9999 ) with confidence 0.9995</td>
<td>37965</td>
<td>17991.783s</td>
</tr>
<tr>
<td>2</td>
<td>( R_{1BBW} ) ( Pr<a href="Monitor.End">&lt;= 35</a> )</td>
<td>( Pr \in [0.990015, 1] ) with confidence 0.99</td>
<td>528</td>
<td>12391.89s</td>
</tr>
<tr>
<td>3</td>
<td>( R_{2BBW} ) ( Pr<a href="Monitor.End">&lt;= 75</a> )</td>
<td>( Pr \in [0.9900061, 1] ) with confidence 0.991</td>
<td>538</td>
<td>13751.28s</td>
</tr>
<tr>
<td>4</td>
<td>( R_{3BBW} ) ( Pr<a href="Monitor.End">&lt;= 75</a> )</td>
<td>( Pr \in [0.9800056, 1] ) with confidence 0.99</td>
<td>263</td>
<td>6362.99s</td>
</tr>
<tr>
<td>5</td>
<td>( R_{4BBW} ) ( Pr[&lt;= 75](Monitor.End and RequestedTorque == 0) )</td>
<td>( Pr \in [0.851567, 0.951396] ) with confidence 0.95</td>
<td>79</td>
<td>3214s</td>
</tr>
<tr>
<td></td>
<td>( Pr[&lt;= 75](Monitor.End and RequestedTorque == 0) ( \implies ) ( \text{ABSBrakeTorque == 0} ) )</td>
<td>( Pr \in [0.998, 1] ) with confidence 0.9999</td>
<td>3797</td>
<td>290362.8s</td>
</tr>
</tbody>
</table>

*Figure 5.6: Analysis results snippet of the BBW Simulink model from UPPAAL SMC. For the complete results, we refer the reader to [39].*
Allocation via Integer Linear Programming

Consider that the mapping solution is represented by a vector of binary matrices \( x = \{ x_1 : 1, ..., K \} \), where \( x_{ij}^k \) represents the mapping of the software component \( q_{ij}^k \) to the computing node \( n_j \in N \).

\[
x^k = \begin{bmatrix}
x_{11}^k & x_{12}^k & \cdots & x_{1K}^k \\
x_{21}^k & x_{22}^k & \cdots & x_{2K}^k \\
\vdots & \vdots & \ddots & \vdots \\
x_{N1}^k & x_{N2}^k & \cdots & x_{NK}^k
\end{bmatrix}
\] (5.1)

The power consumption of a software application that is mapped to computing units \( N' \in N \) is calculated as the sum of the power consumption of each node \( n_k \in N' \), i.e., \( P_{\text{total}}(x) = \sum_{n_k \in N'} P(u_n(x)) \), where \( P \) is the power consumption of a computing node which is linearly proportional to its utilization [62]. The utilization of a computing node is the sum of the utilization of its constituent tasks (or software components). Equation (5.2) computes the utilization of the nodes.

\[
(u_1, ..., u_K)(x) = \sum_{k=1}^{K} \sum_{\tau \in T_{ci}} x_{ij}^k \frac{C_\tau}{P_\tau},
\] (5.2)

where \( T_{ci} \) is the set of tasks which realize the software component \( c_i \) (which is a type for \( q_{ij}^k \) replica), \( C_\tau \) and \( P_\tau \) are the worst-case execution time and period of the task \( \tau \), respectively.

We apply the classical worst-case response time analysis [14] to check the schedulability of tasks sets in each node. The analysis is recursive, and depends on a higher priority tasks set, which is dynamic for different mappings. The end-to-end delay of a chain is calculated for data age (also known as age delay), which is iterative. Thus, we propose a formulation of the timing constraints as logical linear constraints as follows: (i) we identify the software components partitions \( P = 2^C \), where \( p \in P \) can be mapped to a node; (ii) we check the schedulability of each partition in each node \( Y_j = \{ p \in P | \text{isSched}(p, n_j) \} \), where \( \text{sched} \) is a Boolean predicate that returns true if \( p \) is schedulable on \( n_j \). A partition is identified by a sequence of 0s and 1s based on its constituent 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 \( x \), the partition id that is mapped to the computing node \( n_j \) can be computed as

\[
g_j(x) = \sum_{i=1}^{N} 10^{N-i} \max_{1 \leq k < K} x_{ij}^k
\] (5.3)

where the \( \max_{1 \leq k < K} \) function returns 1 if at least one component replica is mapped to \( n_j \) 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,
\[
\bigvee_{p \in Y_j} g_j(x) = id(p),
\]
where \(id(p)\) is a predicate that returns the id of the partition \(p\).

Similar to the tasks deadline constraints, we construct logical constraints that correspond to the valid set of chains. Given \(\Gamma = \{\Gamma_i : i = 1, \ldots, N\}\) chains, the set of possible mappings of each chain \(\Gamma_i\) on a set of \(N\) nodes is \(\Gamma_i^N\). Thus, the set of valid mappings of each chain that satisfy the end-to-end timing requirements are \(Z_i = \{\gamma \in \Gamma_i^N | ageDelay(\gamma) \leq EE_{\gamma}\}\).

The reliability constraint refers to the probability that the application functions by the time \(t\), that is, within the interval \([0, t]\) [42]. Unless the software application is replicated on one or more computing nodes, the reliability requirement is usually unattainable. Therefore, the allocation strategy is basically to replicate sufficient software components to meet the requirement. However, the reliability calculation is not trivial as the replication results in functional inter-dependency of the computing nodes due to the replication of components, in which case, the series-parallel method does not always apply. Instead, we propose an exact method based on state enumeration [59] of the computing nodes, i.e., all the possible configurations of the nodes \(PS\) are enumerated. Then, the reliability of the application is computed as the total probability of those configurations that allow functioning of the application [62].

Allocation via Hybrid Particle Swarm Optimization

Meta-heuristic algorithms do not guarantee optimality of solutions, nevertheless, 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., accommodating more and more software applications, improving battery-life, rather than optimality. In this spirit, metaheuristics 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.

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 [84]. Its hybridization is proven to perform better in many cases [88]. In particular, it is shown to perform better in the tasks assignment problem, that is when hybridized with, e.g., the genetic algorithm [86], the hill-climbing [96], simulated annealing [98], differential evolution [92].

37
compared to the hybridization with genetic algorithms, the hybridization with hill-climbing HCPSO is shown to perform better [96] for the tasks allocation problem to maximize reliability of distributed systems.

In this work, we apply HCPSO and its stochastic version SHPSO to the problem at hand, in order to tackle the PSO stagnation when applied to large and complex 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 shows the pseudocode of the hybrid PSO. Line 3 and 4 compute the personal best and the population (or swarm) best solutions, respectively. For each particle in the swarm, the velocity and position are 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, that is, whenever the interval criterion condition is satisfied.

```
input : PSOparameters, DEparameters
output: Software allocation solution sBest.x
1 Particles $P \leftarrow$ initPSO();
2 while termination criteria do
3    $p_{bst}$ $\leftarrow$ ComputePersonalBest ($P$);
4    $z$ $\leftarrow$ ComputeSwarmBest ($P$);
5    foreach $p \in P$ do
6        computeParticleVelocity ($p$) according to Equation (2.8);
7        computeParticlePosition ($p$) according to Equation (2.9);
8    end
9    if interval criteria then
10       $P \leftarrow$ optimizeUsingDE ($P$);
11       // $P \leftarrow$ optimizeUsingHC ($P$)
12       // $P \leftarrow$ optimizeUsingSHC ($P$)
13    end
14 end
```

**Algorithm 1:** Hybrid PSO Pseudocode.

Validation on the Bosch EMS Benchmark

Our proposed software allocation approaches are validated on different sizes of software applications, i.e., applications with variable numbers of software components, cause-effect chains and degree of replication. The applications are synthesized according to the automotive benchmark proposed by Kramer et al. [54], which contains the timing specifications of runnables and their shares in the AUTOSAR implementation of an engine management system.
(a) Power consumption of nodes.

Figure 5.7: ILP optimization of different sizes of software applications based on the number of software components, cause-effect chains, computing nodes. EMS. Moreover, it shows the activation patterns of cause-effect chains, number of 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 platform.

The ILP approach validation: we have prepared 7 software applications and an execution platform that consists of 8 nodes. The smallest application consists of 4 components and the largest application 10 components. Furthermore, the cause-effect chains vary from 10 to 60, the activation patterns range from 2 to 4, which are consistent with the benchmark specifications.

Figure 5.7 shows the result of the ILP optimization as solved by the CPLEX solver. The algorithm selects the computing nodes with the lower power specifications as more resources are needed as more components are allocated. Note that, the figure legend shows a list of nodes listed in ascending order according to power consumption specifications. Figure 5.7b shows a marginal increase of computation time until the number of components hits 8, which
takes 6.06 sec. Afterwards, it rapidly increases to 30.3 sec and 129.4 sec for allocating 9 and 10 components, respectively. The optimization takes a large amount of time to allocate 15 components, and extremely large computation time to allocate more than 15 components, consequently interrupted manually. For more of the validation results and analysis, please consult Paper D [62], also included in this thesis.

The hybrid PSO approach validation. We have prepared 7 software applications, which are identified by \(\langle cimjgk \rangle\), where \(c, m, g\), respectively, refer to the components, computing nodes and chains, and \(i, j, k \in \mathbb{I}^+\), respectively, refer to the cardinality of \(c, n, g\), respectively, in the application.

As opposed to the ILP approach, in this validation, we have to consider quality of the solutions since the solutions delivered by meta-heuristic algorithms do not guarantee optimality. Furthermore, we evaluate the impact of the proposed approximation algorithm to minimize the overhead of the replication on the optimization.

Figure 5.8 shows that ILP and all except PSO and DEPSO return optimal solutions for the first three problems. However, as the problem size increases, the hybrid PSO with the local search algorithms, i.e. HCPSO and SHPSO, deliver better solutions as compared to DEPSO.

![Figure 5.8: Power consumption of nodes for different instances of software allocation problems by different optimization algorithms.](image)

Figure 5.9 shows that the hybrid PSO algorithms with local search perform the best in the last three problems except HCPSO, which fails to return near
optimal solutions to the largest problem $c_{80g20n60}$ due to extremely large computation time, consequently manually interrupted. Thus, the stochastic version of the hill-climbing algorithm scales well, and its overall quality of the solution can be deemed acceptable as compared to ILP or HCPSO. The analysis of the approximation algorithm that is introduced to lower the overhead of replication on the end-to-end computation has resulted in lower computation time, as expected, while impacting less the quality solutions. For more of the validation results and analysis, refer Paper F [63].

![Figure 5.9: Computation time of the software allocation for different instances of the software allocation problems by different optimization algorithms.](image)

5.6 Included Papers and their Mapping to Contributions and Goals

Paper A


**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**


**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


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


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:** Revisions required.

**Paper E**


**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 was the main driver of the paper. I developed the system model with the guidance of the co-authors, and formulated the optimization problem with the guidance of Hamid Faragardi, implemented the the problem in Java, and collected and analyzed the experimental results. The co-authors gave writing updates, useful ideas and comments on the paper, and specifically: Guillermo Rodriguez-Navas on reliability analysis, Hamid Faragardi on optimization, Saad Mubeen on the timing analysis and Cristina Seceleanu on the objective function and constraints.

**Status:** Published
Paper F


Abstract: It is desirable to optimize power consumption of distributed safety-critical software that realize fault tolerance and maximize reliability as a result, to support the increasing complexity of software functionality in safety-critical embedded systems. Likewise, safety-critical applications that are required to meet end-to-end timing constraints may require additional computing resources. In this paper, we propose a scalable software-to-hardware allocation based on hybrid particle-swarm optimization with hill-climbing and differential algorithms to efficiently map software components to a network of heterogeneous computing nodes while meeting the timing and reliability constraints. The approach assumes fixed-priority preemptive scheduling, and delay analysis that value freshness of data, which is typical in control software applications. Our proposed solution is evaluated on a range of software applications, which are synthesized from a real-world automotive AUTOSAR benchmark. The evaluation makes comparative analysis of the different algorithms, and a solution based on integer-linear programming, which is an exact method. The results show that the hybrid with the hill-climbing algorithms return very close solutions to the exact method and outperformed the hybrid with the differential algorithm, though consumes more time. The hybrid with the stochastic hill-climbing algorithm scales better and its optimality can be deemed acceptable.

Personal Contributions: I was the main driver of the paper. I developed the system model with the guidance of the co-authors, and formulated the optimization problem with the guidance of Hamid Faragardi, implemented the problem in Java, and collected and analyzed the experimental results. The co-authors gave writing updates, useful ideas and comments on the paper, and specifically: Guillermo Rodriguez-Navas on reliability analysis, Hamid Faragardi on optimization, Saad Mubeen on the timing analysis and Cristina Seceleanu on the objective function and constraints.


In summary, the mapping between the included papers and the research contributions is given in Table 5.1, whereas the mapping between the research contributions and the research goals is given in Table 5.2.
<table>
<thead>
<tr>
<th>Paper</th>
<th>RC1</th>
<th>RC2</th>
<th>RC3</th>
<th>RC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper A</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper B</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper C</td>
<td></td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper D</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Paper E</td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Paper F</td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

Table 5.1: *Mapping between the included papers and the research contributions.*

<table>
<thead>
<tr>
<th>Contribution</th>
<th>RG1</th>
<th>RG2</th>
<th>RG3</th>
<th>RG4</th>
<th>RG5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC1</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC2</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC3</td>
<td></td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC4</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: *Mapping between the research contributions and the research goals.*
6. Related Work

In this section, we discuss and compare to the related work on specification and analysis of requirements, formal analysis Simulink models and allocation of software applications.

6.1 Specification and Analysis of Embedded Systems Requirements

Embedded system requirements are captured in different representations, e.g., textual, tabular, graphical, etc. The textual representation can be conveniently classified into two classes: i) controlled natural language (CNL), ii) template-based methods. The syntax and semantics of the CNL 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 rigorous 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 [55], e.g., Attempto Controlled English (ACE) [40], Processable English (PENG) [87], etc. Similar to most computer-processable languages, ReSA has limited syntactic constructions, and allows knowledge representation. However, unlike Attempto or PENG, it is tailored to the specification of embedded systems requirements. On the cons side, it does not support advanced natural language processing features, like anaphra parsing. 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 (boilerplates), which are reusable, recurrent patterns to specify requirements, e.g., CESAR boilerplates [36]. The main drawback of existing requirements boilerplates are: i) the templates are usually too limited therefore not expressive enough, and ii) it is not easy to find the appropriate boilerplate during specification. In this regard, ReSA extends boilerplates with a meta-model that guides a 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 models. The approaches differ mainly in their coverage of the Simulink language, their robustness to formally analyze complex Simulink models, which is crucial for applicability of the proposed methods in industry. In this related work, we focus on techniques that employ model checking due to the extra benefit of a higher degree of automation if compared to theorem proving methods.

Simulink already supports the formal analysis of descriptions via its Simulink Design Verifier (SDV) product. Although there are limited resources that investigate the pros and cons of the product, the studies by Nellen et al. [76] and Florian et al. [58] indicate some limitation of the product, e.g., its inability to specify timed properties (e.g., cause-response LTL properties), degraded performance and less scalability against the SPIN model checker, inability to detect problems caused by race conditions in concurrent processes.

The PlasmaLab proposed by Axel Legay et al. [57] transforms Simulink sample traces into statistical models, which are eventually analyzed by their statistical model checker. Although 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 delivers sample traces, which is the main advantage of the tool.

Other related work relies on exact model checking as opposed to statistical, e.g., Meenakshi et al. [71] propose a transformation into the input language of NuSMV model checker and supports only discrete Simulink blocks. Similarly, Bernat et al. [12] propose a transformation into an intermediate language, followed by compilation of the latter into Common Explicit-State Model Interface (CESMI) specification, which is an input language to the DiVINE model checker. These related approaches are limited in scalability due the use of exact model checking, thus cannot be used on complex and large-scale Simulink models. Other research endeavors propose the transformations of only StateFlow/Simulink models into Bayesian statistical models [100], hybrid automata [67], timed automata [50].

In contrast to the discussed related work, our approach is distinct mainly in three ways: i) we transform the Simulink model into NSTA, which are checked via statistical model checking, using UPPAAL SMC, which tames scalability with respect to large industrial Simulink models; ii) we verify the routines of the Simulink blocks, which are implemented in C, using Dafny (a program verifier developed at Microsoft Research); iii) our approach supports transformations of any type of Simulink blocks, thus it can handle mixed discrete-time and continuous-time Simulink models.

---

https://se.mathworks.com/products/sldesignverifier.html
6.3 Software Allocation Optimization

Different allocation schemes deliver different system performance and therefore efficient software allocation is crucial. Ernest Wozniak et al. [95] proposed a synthesis mechanism for an AUTOSAR software application that can execute over multiple nodes, with the objective of fulfilling timing requirements. In contrast, we consider power consumption and reliability requirements besides timing. Similarly, Salah Saidi et al. [85] 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 network of computing nodes. Ivan Svogor et al. [93] 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 complex timing analysis especially if data age constraints are considered [73]. In addition AHP scales for at most 10 components. As opposed to the previously mentioned related work, we considered a system model with multi-rate tasks, and data age constraints on the end-to-end communication of the tasks. Moreover, we consider fault-tolerant application, i.e. by replicating software components, to maximize the reliability of the application.

Consequently, our allocation approach considers both reliability requirements as well as end-to-end timing ones as constraints to be satisfied. On a different front, there exists research focusing on reducing energy consumption in distributed systems based on dynamic voltage scaling [10], redistribution of computation load in datacenters via “bio-inspired algorithm” [11], which are applicable during runtime but does not consider reliability and end-to-end response time. And other works apply task consolidation to minimizing computational nodes [35][33], however do not consider power consumption as the objective function.
7. Conclusions and Future Work

In this thesis, we have proposed formal methods and optimization techniques for the assured and efficient design of safety-critical embedded systems. We have developed a domain-specific requirements specification language, called ReSA, which is tailored to embedded systems, and applied SAT-based and ontology-based techniques to improve quality of specifications. We have also proposed an approach for formally modeling hybrid Simulink models via the stochastic timed automata formalism. Our approach is based on an automated pattern-based, execution-order-preserving method to transform large-scale Simulink models into networks of stochastic timed automata. The automata are analyzed for function and end-to-end timing requirements via UPPAAL SMC [22]. For resource-constrained design in general and low-battery embedded systems in particular, we have proposed efficient software allocation methods via integer linear programming and hybrid particle optimization on a network of heterogeneous computing nodes, with respect to power specification, failure rate and processor speed. During the allocation, the timing and the reliability constraints are satisfied by considering the worst-case response time, age delay, and exact reliability analysis in the context of fault tolerance ensured by node redundancy.

Our solutions are validated on various industrial automotive use cases and one benchmark. The ReSA language is validated on the Adjustable Speed Limiter (ASL) use case, for which we are able to express around 90% out of 300 requirements, which are mostly conditional statements. However, the 10% of ASL requirements contain quantifiers, temporal and timed properties, which are difficult to handle via the SAT-based approach. The SAT-based approach as opposed to the ontology-based approach scales well, moreover, it is fully automated and is also integrated into the EATOP tool chain at VGTT. However, the analysis with the SAT-based approach is shallow due to abstraction of clauses whereas the ontology-based approach is rigorous as it operates at the lexical level but it scales less.

Our proposed formal analysis of Simulink models is validated on the BBW system that contains 320 Simulink blocks, to which it scales well and partially on the ASL system, on which the transformation could not handle some of the Simulink blocks, hence the generated model is not immediately analyzable. However, the limitations are due to the implementation of our tool SIMPPAAL that we also propose in this thesis, which can be improved and limitations removed by future work.
Our proposed software allocation approaches are validated on the AUTOSAR benchmark produced by Bosch, which consists of several synthetic applications. The results show that the ILP approach scales to a medium size of application software, i.e., not more than 15 software components and 60 chains. In contrast, the hybrid particle swarm optimization scales well to very large applications, consisting of 80 software components and 60 chains per application. Of the metaheuristic algorithms evaluated in our experiments, the hybrid PSO algorithms with hill climbing provided better quality solutions next to ILP on small problems, and performed best on larger problems as compared to the classical PSO and DE algorithms, as expected. In particular the hybrid PSO with the stochastic hill climbing algorithm outperforms the rest.

As lines of future work, we foresee the following: (i) implementation of the ontology-based requirements analysis using open-source lexical databases, e.g., WordNet; (ii) generalization of the statistical model checking to any data-flow programming paradigms; (iii) extension of the software allocation problem to include dynamic synthesis of tasks; (iv) extension of the software allocation techniques to also address the dynamic power and dynamic configuration of software components.
8. Bibliography


