UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Modeling and Co-Design of Multi-domain Cyber-Physical Systems

Permalink
https://escholarship.org/uc/item/36b3j20g

Author
Wan, Jiang

Publication Date
2018

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Modeling and Co-Design of Multi-domain Cyber-Physical Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Jiang Wan

Dissertation Committee:
Associate Professor Mohammad Al Faruque, Chair
Professor Fadi Kurdahi
Professor Rainer Dömer

2018

Chapter 3 © 2015 IEEE
Chapter 4 © 2017 IEEE
Chapter 5 © 2017 IEEE
Chapter 7 © 2015 IEEE
All other materials © 2018 Jiang Wan

DEDICATION

This thesis is dedicated to my parents for their support and love. This work is also dedicated to my advisor for his guidance and support during my challenging research life.

TABLE OF CONTENTS

Page

LIST OF FIGURES **vi**

LIST OF TABLES **ix**

ACKNOWLEDGMENTS **x**

CURRICULUM VITAE **xi**

ABSTRACT OF THE DISSERTATION **xiii**

1 Introduction **1**
- 1.1 CPS Modeling and Design . . . 1
- 1.2 Dissertation Objective . . . 3
- 1.3 Dissertation Outline . . . 4

2 Background and Related Work **5**
- 2.1 Modeling of Functional Requirements . . . 7
- 2.2 Modeling of Non-functional Requirements . . . 10

3 CPS Design Case Study **12**
- 3.1 Introduction and Related Work . . . 12
- 3.2 Our Contributions . . . 15
- 3.3 IEC-61131-3 Based Soft-PLC . . . 15
- 3.4 MBD of TTDRE for Soft-PLC Platform . . . 17
  - 3.4.1 Design Methodology . . . 17
  - 3.4.2 Executable Task Generation . . . 18
  - 3.4.3 Automatic Synthesis of Real-time Preemptive Scheduler for the Executable Tasks . . . 22
- 3.5 Validation Framework . . . 24
  - 3.5.1 Our Framework . . . 24
  - 3.5.2 Communication and Synchronization . . . 25
- 3.6 Results AND Evaluation . . . 26
- 3.7 Conclusion . . . 27

4 Functional Model-based Design Methodology **30**
4.1 Introduction . . . 30
4.2 Background and Related Work . . . 33
4.3 Functional Model compiler (FMC) . . . 36
4.3.1 Feedback Function . . . 39
4.3.2 Architecture Model Synthesis Step (**Step 1**) . . . 40
4.3.3 Simulation Model Synthesis Step (**Step 2**) . . . 48
4.3.4 Refinement Process . . . 50
4.4 Case Study . . . 55
4.4.1 Synthesis of Architecture Models . . . 56
4.4.2 Synthesis of Simulation Models . . . 56
4.4.3 DSE Analysis using FMC . . . 58
4.4.4 The Effectiveness of Feedback Function . . . 59
4.4.5 Analysis of the Refinement Process . . . 60
4.5 Conclusion . . . 62

5 CPS Co-Design at the Functional-Level **64**
5.1 Introduction . . . 64
5.2 Motivational Case-Study . . . 68
5.3 Cyber-Physical Co-Design . . . 71
5.3.1 Definitions . . . 72
5.3.2 Co-Design Flow . . . 74
5.3.3 Control-Physics Partition . . . 75
5.3.4 Cost Metrics . . . 77
5.3.5 Cyber-Physical Co-Synthesis Algorithm . . . 79
5.4 Experiments and Case-Study Analysis . . . 83
5.4.1 Analysis of the Synthesis Process . . . 84
5.4.2 Simulation and Validation . . . 87
5.5 Related Work . . . 88
5.6 Conclusion . . . 91

6 System Level CPS design using Non-Euclidean Training method **92**
6.1 Introduction . . . 92
6.2 Related Works . . . 94
6.3 Contributions . . . 96
6.4 SGCNN Architecture . . . 97
6.4.1 Neighbor Nodes Aggregation . . . 97
6.4.2 Subgraph Convolution Kernel . . . 100
6.5 Graph Pooling Algorithm . . . 102
6.6 Experimental Results . . . 104
6.7 Conclusion . . . 107

7 Security-Aware Functional Modeling of Cyber-Physical Systems 108
7.1 Introduction and Related Work . . . 108
7.2 Our Novel Contributions . . . 110
7.3 Security-Aware Functional Model . . . 110
7.4 Design Automation Tool . . . 111
7.5 Experimental Results . . . 116
7.6 Conclusion and Future Work . . . 118

8 Securing Wireless Communication in Automotive CPS 119
8.1 Introduction . . . 119
8.1.1 Motivation . . . 120
8.1.2 Security Challenges for V2X Communication . . . 121
8.1.3 Related Works . . . 124
8.1.4 Overview of Contributions . . . 125
8.2 System Model . . . 127
8.2.1 Wireless Communication Model . . . 127
8.2.2 Attack Model . . . 130
8.2.3 Security Strength Evaluation . . . 131
8.3 Pre-Shared Key Generation . . . 132
8.3.1 Algorithm Pseudocode and Description . . . 133
8.4 Pre-Shared Key Length Optimization . . . 135
8.4.1 Scenario Mapping . . . 136
8.4.2 Algorithm Pseudocode and Description . . . 139
8.5 Cryptographic Key Derivation . . . 141
8.6 Simulation Results . . . 143
8.6.1 Pre-Shared Key Generation . . . 143
8.6.2 Pre-Shared Key Length Optimization . . . 146
8.7 Experimental Results . . . 147
8.7.1 Remotely Controlled Car Environment . . . 148
8.7.2 Automobile Environment . . . 151
8.8 Evaluation . . . 152
8.8.1 Security Comparison . . . 153
8.8.2 Performance Overhead Comparison . . . 154
8.8.3 Code Size Overhead Comparison . . . 155
8.9 Conclusion . . . 156

Bibliography 157

LIST OF FIGURES

Page

2.1 System Level CPS design challenges 6

3.1 Traditional MBD methodology of industrial TTDRE system 14
3.2 IEC 61131-3 language: (a) language structure; (b) and (c) examples of IEC 61131-3 languages 16
3.3 Our proposed MBD methodology and complete tool-chain (the highlighted parts are our contributions and rest of the parts are off-the-shelf tools/processes) 17
3.4 Presented configuration knobs 20
3.5 Our proposed SILS framework 23
3.6 Application scenario of the factory robot-arm use-case 24
3.7 Wrist controller analysis using our tool-chain 25
3.8 Comparison of our scheduler to Beremiz scheduler 28
3.9 SAD between the results MILS and SILS 29

4.1 The overview of the FMC design flow. 38
4.2 "`Feedback`" function semantics is the reuse or returning of material, energy, or signal flow to a predecessor function. Syntactically, it can be visualized as a flow from a `successor` to a `predecessor` function. 40
4.3 A context tree for the "Convert" function of the automotive system. The flows are separated into *supplying* and *receiving* flows. The supplying flow is linked with the input flow directly connected to the function, while the receiving flow is the output flow. By following the context tree, the algorithm is able to infer the correct components needed to realize the function. For example, following the highlighted flows, a combustion engine component is selected. Moreover, a catalytic converter may be selected if both the supplying and the receiving flows are gas; a human driver may be selected if the receiving flow indicates a human energy; ECUs may be selected if the flows are signals. . . 42
4.4 Extraction of functions from simulation components. 52
4.5 Structural analysis on simulation components reveals additional domains and the energy, material, signal transformations between them. 53
4.6 An example functional model of an automotive power-train, associated requirements, and architectural constraints. 54
4.7 Architecture templates for the engine block. 55
4.8 Synthesized `Series Hybrid` architecture simulation model. 57

4.9 Battery's SOC among the `Series Hybrid`, and `Parallel Hybrid` architectures on the New European Driving Cycle (NEDC). 58
4.10 Fuel consumption comparison between the `ICE`, `Series Hybrid`, and `Parallel Hybrid` architectures. 59
4.11 Synthesized simulation model for the `Electric` architecture. 60
4.12 Comparison of the fuel consumptions between the synthesized simulation models with/without "Feedback". 61
4.13 Comparison of emissions between the simulation models with/without "Feedback". 61
4.14 Bottom-up refinement of a functional model. 62
4.15 Number of functions in the refined functional models. 62

5.1 Two example architectures for the CPS functional model of the automotive steering system. 68
5.2 Performance and power consumption comparisons for the software-centric (SBW) and electro-mechanical-centric (HPS) automotive architectures. . . . 69
5.3 Proposed co-design methodology for automotive CPS. The right half of the figure describes the detailed steps of the proposed co-design methodology. The left half of the figure gives a synthesis example of a steering system (CPS functional model → Multi-domain architecture model → Simulation model). 71
5.4 Control-physics partitioning of the steering system. 76
5.5 The CPS functional model for parts of an automotive. 83
5.6 Comparison of the cyber domain architectures between DBW and DBW (multi-core). 86
5.7 The input signals of a corner-turning scenario. 87
5.8 Performance comparison for various automotive architectures. 88
5.9 Power consumption comparison for various automotive architectures. 88
5.10 Tradeoff analysis between reliability and performance costs. 89

6.1 SGCNN Architecture 97
6.2 Neighbor Nodes Aggregation Example: Yellow node is a simple representation of a subgraph $\mathcal{G}_t$; Blue nodes represent length 5 paths; Green nodes represents length 4 paths. 98
6.3 Attribute Matrix Example 101
6.4 Convolution Kernel Example: Given input graph having 4 vertices, a 3-by-3 convolution kernel is applied. 4 convolution candidates are generated and the convolution results a new graph with 4 vertices 102
6.5 Subgraph Convolution Layer Architecture 103
6.6 Example of a subgraph constructed from GrabCAD model 105
6.7 Loss value during the training process 106
6.8 Accuracy results during the training process 106

7.1 Security-aware functional level model of automotive. 112
7.2 Our proposed design automation tool. 113
7.3 Implementation of the proposed design automation tool. 114

7.4 Simulation results without attack models. 117
7.5 Simulation results with the proposed attack models: (a) fuzzy attack, (b) interruption attack, (c) man-in-the-middle attack, (d) replay attack, (e) overflow attack, (f) down-sampling attack. 118

8.1 Examples of V2V and V2I Applications 120
8.2 Overview of the Proposed Scheme for Secured Automotive Wireless Communication 127
8.3 Physical Layer PSK Generation for a V2V Scenario 133
8.4 Simulation Results of Our Key Generation Overhead 144
8.5 Simulation of Generating Two Secret Keys at the Same Time 145
8.6 Optimized Key Length for Fast Fading Model 147
8.7 Optimized Key Length for Slow Fading Model 147
8.8 Optimized Key Generation Time for Fast Fading Model 148
8.9 Optimized Key Generation Time for Slow Fading Model 148
8.10 RC Car Experiments Setup 149
8.11 Collected Samples of RSS Values from the RC Car Experiments 150
8.12 Real World Experiment Using Phones and Laptops 151
8.13 Collected RSS Values from Real World Experiment 152
8.14 Experimental Key Generation Time at Different Speeds 152
8.15 Estimated Average Min-Entropy Results Comparison 154

LIST OF TABLES

Page

3.1 Timing information of robot-arm control system 26

5.1 Comparison of five synthesized architectures 84
5.2 Example list of the components in the component library 85
5.3 Cost results for various automotive architectures 85

8.1 Comparison of Existing Cryptographic Algorithms 123
8.2 Setup For Our Key Generation Algorithm 143
8.3 Correlations of the Generated Keys 145
8.4 Scenario Mapping Data 146
8.5 Generated 64-bit Keys from the RSS Values 150
8.6 Performance and Code Size Overhead Comparisons Results 155

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Mohammad Al Faruque, for giving me support over the years. I am grateful for his guidance during these years. He is incredibly organized and a great researcher. Under his guidance I have gained the skills of doing scientific research writing academic papers, which are invaluable tools to have as my career moves forward. I will remember my time in the lab and these outings very fondly.

I would like to recognize the current and former members of the AICPS lab who have all contributed to the progress I have made. Over the beginning of my research years, Dr. Haeseung Lee has been working with me in this lab together through those tough days for starting any challenge research topics. I would like to acknowledge Dr. Korosh Vatanparvar for his help on sharing his time and skills to help me on preparing many papers and this thesis. I also wanted to acknowledge Sujit Rokka Chhetri and Anthony Lopez for the inspiring talks and fun time we have been spent together. I would like to thank all the other current or former members, Nafiul Rashid, Sina Faezi, Hsin-Wei Hung, Hsinchung Chen, and Ahmad Albaqsami, they are a wonderful group of people who I have enjoyed working with and learning from.

I would also like to thank Dr. Arquimedes Canedo, for all his great ideas which he has shared with me generously. Without his help, I would not be able to get practical knowledge from the industry.

I thank IEEE and ACM for the permission to include the content of my dissertation, which contains a portion that of it was originally published in transactions and conference papers.

Finally, I would like to thank my friends and family for their continued support and encouragement.

CURRICULUM VITAE

Jiang Wan

EDUCATION

Doctor of Philosophy in Computer Engineering **2018**
University of California, Irvine *Irvine, California*

Master in Electrical and Computer Engineering **2014**
University of California, Irvine *Irvine, California*

Bachelor of Science in Microelectronics **2008**
Shanghai Jiao Tong University *Shanghai, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2012–2018**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2016–2018**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Physical Layer Key Generation: Securing Wireless Communication in Automotive Cyber-Physical Systems — **August, 2017**
ACM Transaction on Cyber-Physical Systems

Cyber-Physical Co-Design at the Functional-Level for Multi-Domain Automotive Systems — **March, 2015**
IEEE Systems Journal

Functional Model-based Design Methodology for Automotive Cyber-Physical Systems — **December, 2014**
IEEE Systems Journal

REFEREED CONFERENCE PUBLICATIONS

Exploiting Wireless Channel Randomness to Generate Keys for Automotive Cyber-Physical System Security — **April, 2016**
International Conference on Cyber-Physical Systems

Security-Aware Functional Modeling of Cyber-Physical Systems — **2015**
IEEE International Conference on Emerging Technology & Factory Automation

Model-Based Design of Time-Triggered Real-time Embedded Systems for Industrial Automation — **2015**
IEEE International Conference on Emerging Technology & Factory Automation

Functional Modeling Compiler for System-Level Design of Automotive Cyber-Physical Systems — **2014**
ACM/IEEE International Conference on Computer-Aided Design

ABSTRACT OF THE DISSERTATION

Modeling and Co-Design of Multi-domain Cyber-Physical Systems

By

Jiang Wan

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2018

Associate Professor Mohammad Al Faruque, Chair

Cyber-Physical Systems (CPS) are integration of computation and physical processes connected through networks. The high complexity of cross-domain engineering in combination with the pressure for system innovation, higher quality, time-to-market, and budget constraints make it imperative for engineers to use integrated engineering methods and tools for CPS design. However, existing computer-based engineering tools are mainly focused on a particular domain and therefore it is challenging to perform system-level analysis for CPS due to the difficulty of knowledge integration from different domains. This thesis studies the problem in the modeling of cross-domain CPS. Problems in the modeling of both functional and non-functional requirements during CPS design are explored and a functional model-based approach is proposed for high level CPS modeling. Moreover, targeting the security requirement in CPS, which is one of the key non-functional requirements in CPS, physics-based models and solutions are proposed in this thesis.

Chapter 1

Introduction

1.1 CPS Modeling and Design

A Cyber-Physical (also styled Cyber Physical) System (CPS) is a system that is controlled or monitored by computer-based algorithms, tightly integrated with the physical world and networks [139]. In CPS, the cyber domain components are control algorithms typically implemented by computer-based software, while the physical domain processes refer to physical behaviors in any other engineering domains which may be controlled by cyber domain components. Physical processes and software components are deeply intertwined through mechanisms like sensors, actuators, and networks [139]. Examples of CPS include smart grid, autonomous automobile systems, medical monitoring, process control systems, robotics systems, and automatic pilot avionics [53, 103].

Traditionally, controlling the physical processes by computer systems has already been deeply studied and widely deployed in many engineering areas. This type of computer system is also known as embedded system. Embedded system is typically a real-time computer system within a larger mechanical or electrical system [103]. In the narrow sense, CPS can be

considered as this larger system in the domain of mechanical or electrical engineering (e.g. mechatronic system). Although it is correct to consider this type of larger system (i.e., mechatronic system) as a CPS, there are many other concepts in a CPS not covered by it [53, 103].

To clearly explain the concepts of a CPS, I would like to claim that a CPS is a system of systems from multidisciplinary engineering domains connected through networks. Thus, the study of CPS should remain at the system level and try to utilize the knowledge from multiple engineering disciplines to develop the whole system [53, 103]. Naturally, the major challenge is that the knowledge in various engineering disciplines are quite different and very difficult to integrate.

In the engineering field, modeling is the most common and practical method to abstract knowledge for system development purposes. Models are typically easy to be defined, possible to be analyzed using scientific methods, and capable of being quantified and simulated. System engineers rely on models to validate both functional and non-functional requirements (e.g. reliability, security). Ideally, a model should be perfectly predictable and as accurate as possible to real implementation. However, different models in different fields at different levels of abstraction may have their own properties which require careful design and sometimes trade-offs [86]. Moreover, more problems will arise when they are integrated to develop a CPS. This dissertation will present a study of some of these problems in CPS modeling and design.

In general, models at high level are more predictable in terms of their functional and non-functional performance. For example, in cyber world, the software model at programming language level such as C/C++, and Java is usually very predictable in terms of its functionality. However, most of the software programming languages lack of timing information, thus when the software models move to lower level (e.g. Operating system level or Instruction Set Architecture level), some of the non-functional requirements may become not perfectly

predictable (e.g. timing, reliability). This problem becomes very critical in CPS, since the non-functional requirements are equally important as functional requirements in CPS development.

Another problem is that integrating models from different domains also make the whole CPS model difficult to meet its functional and non-functional requirements. This is not a new problem in engineering domain. For example, many systems in domains such as automotive electronics, home appliances have already been studied and developed in practice. However, the cyber components and physical components in most of such systems are developed separately. For example, in most cases, the embedded software engineers are usually provided a model of physical components or the real implemented physical systems for their software development. Thus, engineers from different domains are not able to efficiently explore system designs collaboratively and analyze trade-offs between cyber and physical world [34]. In order to advance the CPS design automation at system level, models/languages in terms of design practice that is common to all the involved disciplines in CPS are being studied [34, 35, 167].

1.2 Dissertation Objective

The objective of the research is to study the problems in system level design of CPS and explore solutions. It is intended that the research findings will contribute to the new modeling and design methodologies to achieve functional and non-functional requirements of CPS during the design stage. The research studies examples of CPS design such as automotive systems to evaluate the contribution. These above aims raise the following core project objectives:

1. To identify the problems in the current CPS design process.

2. To develop new modeling and design methodology for system level CPS design.

3. To study and solve real problems in practical CPS design projects using my research findings.

1.3 Dissertation Outline

This dissertation is organized as follows. Chapter 1 introduces the objective of this dissertation. Chapter 2 aims to explore the background and specific problems in CPS modeling and design through reviewing the existing research. Chapter 3 presents a case study on CPS design using a model based design methodology. Chapter 4 and 5 describe the proposed new functional model for CPS modeling. Chapter 6 propose a new functional lifting method though non-euclidean training algorithm. Besides modeling the functional requirements, chapter 7 studies modeling the security of CPS, which is one of the non-functional requirements; and chapter 8 presents a case study through a project of physical layer key generation in wireless communication in automotive CPS.

Chapter 2

Background and Related Work

Cyber-Physical Systems (CPSs) are hybrid (continuous and discrete dynamics), heterogeneous (component variations), multi-domain systems consisting of various embedded systems, and physical processes connected through communication networks. The complexity in the modern CPS design is attributed, for the most part, to the software integration [38, 105, 137] during the engineering stage. For example, today, 90% of all the automotive innovations come from the electronics and software; and 40% of the development costs come from the Electronic/Electric (E/E) systems [62].

In the semiconductor industry, various Hardware/Software co-design methodologies and design automation tools [66, 72] have been proposed for identifying the optimized architecture for the given system objectives. Moreover, the co-design methodology has also been shown to be applicable for the multi-domain physical systems consisting of various physical domains such as electrical, mechanical, hydraulic, pneumatics, etc. [161]. In recent years, with the increasing interactions between the electronic-domain (control/computation/communication) and the physical domain, design space exploration across software-integrated multi-domains (continuous and discrete dynamics) has been considered as the grand research challenge for

designing the complex CPS [164].

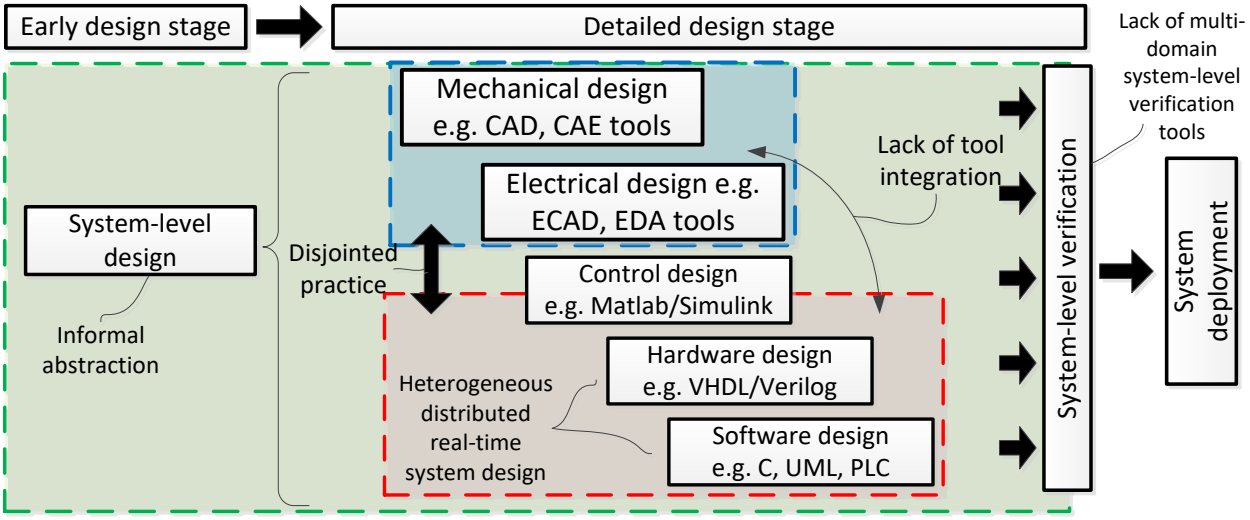


Figure 2.1: System Level CPS design challenges

As shown in Figure 2.1, the critical path in a complex CPS design is the *detailed design* phase [7, 93, 173], where precise engineering specifications are created by the domain experts. Thus, mechanical engineering is supported by Computer-Aided Design (CAD) and Engineering (CAE) tools; electrical engineering by Electronic Design Automation (EDA) and wire harness design tools; control engineering by LabView [100], LMS [108], Modelica [123] and Matlab/Simulink [160]; and software engineering by UML [117] and in-house software development environments [25]. Unfortunately, because of the incompatibility between different domains, these tools cannot be easily and effectively combined to perform system-level analysis and simulations [53]. This situation has led to a serious problem because the CPS's complexity keeps increasing but the design tools cannot answer simple but important questions such as estimating the impact of a change in the transmission architecture in the overall fuel economy of an automobile [179]. Model-Based Design (MBD) methodologies and tools [61] are becoming more popular as they allow both cyber and physical components to be designed, analyzed, implemented, and verified against system-level and high-fidelity multi-disciplinary systems in model-, software-, and hardware-in-the-loop simulations.

MBD is a mathematical and visual method of addressing problems associated with designing

CPSs [132]. Traditionally MBD for CPS involves a sequential and independent development of the physical plant, and then the control system [28, 119]. This is because a control algorithm is typically highly tuned for a specific dynamic behavior of the physical plant. *Model-In-the-Loop Simulation* (MILS) is performed to verify the control algorithm at the model level. Therefore, an accurate physical plant model is constructed prior to the development of the control algorithms. The major problem of the sequential design process is that it requires a longer design time–including both the design time of the physical plant and the control system. Moreover, another important drawback is that the whole system can only be tested or verified *after* the integration of both the real physical plant and the embedded control code, any unexpected change to the physical plant or the control system requires re-execution of all the simulation/testing-processes of the entire system.

To address these challenges, I present this thesis which is made up of the studies I have performed on CPS design with MBD. Specifically, I studied some of the challenges in the modeling of both functional and non-functional requirements.

2.1 Modeling of Functional Requirements

As mentioned, the software integration in the CPS design is performed after the physical architecture has been decided. This is not an optimal method for various domains and is a counter-intuitive methodology for the design of CPSs because, in principle, CPSs are intended to be tight integrations of "cyber" and "physical" functions. In my study, the state-of-the-art CPS design methodologies should be revisited in such a way that the functionality of software is validated as early as possible, and in parallel with the design of the rest of the system. This means that new design tools [48] should provide the CPS design capabilities to fully exploit the functionalities of both cyber and physical components.

Multi-domain design is addressed in existing CPS design methods, but most of them are limited in targeting the cyber parts or the physical parts separately. There exists no method that is capable of providing a unified abstraction of the system that could explore the software-integrated multi-domain design space. Description languages such as Architecture Analysis & Design Language (AADL) described in [60] is limited to describing the architectures for software. Modeling languages such as Modelica [171] or Simscape [112] are mainly proposed for modeling the multi-domain physical components. The LTL specifications proposed in [187] are specialized for control algorithms. For example, Pessoa [114] is a tool capable of synthesizing the embedded controllers from the control specifications. Work in [8] has demonstrated the importance of multi-domain design for a CPS in the system-level. However, [8] does not provide a solution for automatically generating architecture models or detailed designs from the system level models.

Furthermore, Model-Based Designs (MBD) [53] and, Concurrent Engineering (CE) [169] are proposed for reducing design time and exploring the design space of multi-domain CPS designs. However, all these works mainly rely on knowledge of the domain experts to make the decisions manually during the modeling stages. Another methodology in [134] named Multi-View Model (MVM) is proposed for designing multi-view systems. Compared to our high-level synthesis methodology, this work focuses on describing the relationship between multi-views (e.g. control view, software view, architecture view etc.) of a design. Besides the design space exploration for architectures, the scheduling of communication and computation is another major research area in the CPS co-design. For example, works presented in [67, 69, 148, 193] provide various methodologies for control-communication co-design of the CPS.

During the early design stage (aka the concept design stage)[1], detecting and fixing errors is 5X to 10X less expensive and problematic than in the latter detailed design stages [61]. Therefore, it is highly desirable to validate CPS design as early as possible. However, the

[1]In systems engineering, the early design stage identifies the overall functionality and structure/architecture of a new system or product [35, 92].

existing CPS design tools stay at the domain-specific levels [23, 53, 140] and this inhibits the possibility of appropriately handling the cross-cutting concerns that characterize CPS. Component designers are deeply specialized in a particular domain (electronic, mechanical, chemical, etc.) and therefore are required to make the detailed design within a subsystem with their "*vertical*" know-how. Another type of designers, referred to as system designers or architects, integrate these detailed designs in a well defined architecture using their "*horizontal*" know-how that spans across disciplines. Notice that a CPS has a very large design space and therefore a control strategy may be implemented in different domains including continuous, discrete, and hybrid. For example, the early design of an automotive requires an experienced designer with a broad multi-domain knowledge to manually make decisions such as: should the control be implemented in a traditional continuous domain architecture or in an E/E hybrid-domain architecture? If E/E architecture is selected, what will be the minimum cost for the digital hardware implementation? What are the additional costs of sensors and actuators? Are additional continuous-domain electro-mechanical components required for implementing this control strategy?

In a CPS, advanced functionality is achieved through the tight integration of physical processes and the associated control systems. A physical process takes some amount/types of energy and/or materials as inputs to produce the expected amount/types of energy and/or materials as outputs. How and when the energy and/or materials are consumed and produced is decided by a connected control system. For example, an engine has a physical process that converts chemical energy contained in the fuel into mechanical energy and hot gases. Moreover, how much and when the fuel is combusted is determined by the force that the driver applies to the accelerator pedal. In this example, the driver is the controller and the accelerator system is the actuator. Furthermore, if the accelerator system takes the control signals from the Electronic Control Units (ECUs) within the E/E architecture of the automotive, a more accurate control of the engine may be provided. This ECU-integrated accelerator E/E system is referred to as "Throttle-By-Wire" technology, and it can increase

the energy efficiency by up to 20% [105].

2.2 Modeling of Non-functional Requirements

Non-functional requirements such as performance, reliability security, power, price, and time-to-market is of equal importance as functional requirements in CPS design [34]. In this thesis, my studies mainly focused on the security requirements in CPS design.

The increasing amount of software and communication networks in CPS deployed in many industries is also making them more vulnerable to cyber attacks [91, 141]. One easy-to-relate example is automobile design; over the years it has shifted from being mechanical-centric to software-centric [35]. It has been shown that different types of attackers and attack scenarios such as through wireless access to the car, through physical access to a car, and through access to the in-vehicle network (tampering Electronic Controls Units (ECU)) may happen in the real world scenarios [40, 94, 149, 165]. An important observation is that even though systems are becoming software-centric, attacks can be physics-based. In [158], the authors have implemented an ABS spoofer that could physically attack the wheel speed sensor so that the functionality of the Anti-lock Braking System (ABS) is completely distorted. These principles can be applied to other complex systems that control critical infrastructure.

Unfortunately, there is currently a gap between the design automation tools and cybersecurity. This thesis advocates the development of new methodologies and tools to support designers, as early as possible during the design cycle, to identify and solve cybersecurity problems in complex systems [96]. In [182], an MBD method to assess the security of CPS is proposed with four architecture-level attack models. Authors in [128] have discussed an MBD technique to quantify the security metrics at the early design stage. To achieve the proposed metrics, a security modeling method is proposed in this paper [128]. Furthermore,

graph-based modeling methods are used in many security problems. Authors in [190] have proposed a systematic method for analyzing cyber-attacks on CPS based on an extended Data Flow Diagram (xDFD) approach. Existing work in [126] has proposed an attack-tree based approach for system-level security design. However, most of the modeling methods used in the existing works are limited to modeling the cyber domain software for security analysis.

Chapter 3

CPS Design Case Study

3.1 Introduction and Related Work

This chapter presents a case study on industrial CPS. The project is conducted on a *Time-Triggered Distributed Real-time Embedded* (TTDRE)[1] systems using a novel *Model-Based Design* (MBD) methodology and associated tool-chain. Our tool-chain automatically synthesizes software for the industrial TTDRE systems. A *Software-In-the-Loop Simulation* (SILS) framework integrated into our tool-chain helps reduce the design iterations.

The TTDRE systems are real-time, networked embedded systems that interact with the physical world using the time-triggered architecture [68, 109]. Today, industrial automation is experiencing a major paradigm shift thanks to the innovations in the semiconductor and software industry that make the TTDRE systems faster, more energy efficient, and reliable [30, 31, 59]. Unfortunately, one of the major problems with a TTDRE system is that once it is deployed, it becomes challenging to extend it. Designers may not have the freedom to influence the timing behavior without reducing the control performance (QoC). For example,

[1] IEC 61131-3 specific *Programmable Logical Controllers* (PLC) are used as the de-facto standard to build industrial TTDRE systems [31].

adding extra software code to a component in a TTDRE system may increase its execution time requiring a re-evaluation of the whole scheduling policy. Extending a TTDRE system will, in most cases, involve a complete re-design of the control system. To solve this problem, MBD [110, 119, 153] has already been seen to be a very promising solution for designing and verifying the industrial TTDRE systems because any modification of a system may be quickly addressed and verified in a higher abstraction. Moreover, hardware/software may be generated from the model using *Computer-Aided Design* (CAD) tools quickly and automatically.

Traditional MBD for the TTDRE systems involves the development of the physical plant and the control process in a separate and sequential manner [28, 119] (see Figure 3.1). This sequential process exists because a control algorithm is highly tuned for a specific dynamic behavior of the physical plant. Therefore, an accurate physical model is constructed before the control algorithms are developed. However this traditional system design approach has an important drawback. Because the control algorithm is fixed to a particular physical plant, any unexpected change to the plant or the controller requires a re-design and re-test of the entire system. To solve this problem, we are proposing a novel MBD methodology to design and verify the physical plant and the control process concurrently.

The industrial TTDRE systems are typically implemented on *Programmable Logical Controllers* (PLCs) programmed using IEC 61131-3 language (see Section 3.3). In [86, 170], MBD methods of PLC application development are discussed. Similarly, Simulink [160] is a commercial MBD tool capable of modeling and simulating TTDRE systems according to a synchronous reactive *Model of Computation* (MoC). *Simulink PLC Coder* [5] is used to automatically generate IEC 61131-3 code for Simulink subsystems. However, there exist the following major limitations for all the above mentioned academic and industrial MBD design methodologies: 1) serialization of design steps due to lack of industry practice and tools, 2) the state-of-the-art *Simulink PLC Coder* [5] tool can only generate non-executable *Function Blocks* (FBs). An FB is an IEC 61131-3 specific semantic (for details see Section

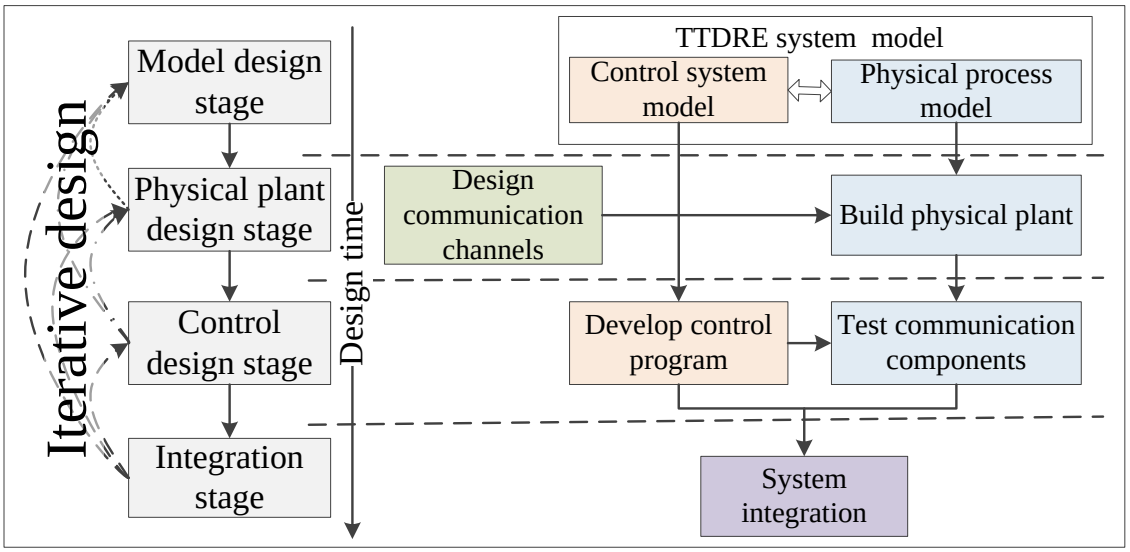


Figure 3.1: Traditional MBD methodology of industrial TTDRE system

3.3) for Soft-PLCs[2], 3) minimum support for design-space exploration of the concept-design phase [55], 4) lack of tool-chain to perform rapid SILS or *Hardware-In-the-Loop Simulation* (HILS) (see details of SILS/HILS in Section 3.5) for the developed TTDRE systems on a target Soft-PLC. Works in [63, 120] have proposed co-simulation solutions. However, these tool-chains do not support simulation in real-time.

To solve these problems, this paper proposes a MBD methodology for TTDRE systems and a complete tool-chain integrating off-the-shelf domain-specific tools and our proposed interfaces & algorithms. Using our tool-chain, designers may automatically generate executable IEC 61131-3 programs with timing (*Worst Case Execution Time* (WCET) [184] and schedulability) analysis for a target Soft-PLC platform from a high level model.

The rest of the paper is organized as follows: In Section 3.2, we present our novel contributions. In Section 3.3, we introduce various preliminaries of PLCs. In Section 3.4, our novel MBD method for the TTDRE system targeting Soft-PLC is described. Section 3.5 discusses our proposed SILS framework. Experimental results are discussed in Section 3.6 with Section 3.7 concluding the paper.

[2]Soft-PLCs or PC-based controllers are control programs running on personal computers.

3.2 Our Contributions

Our novel contributions are as follows:

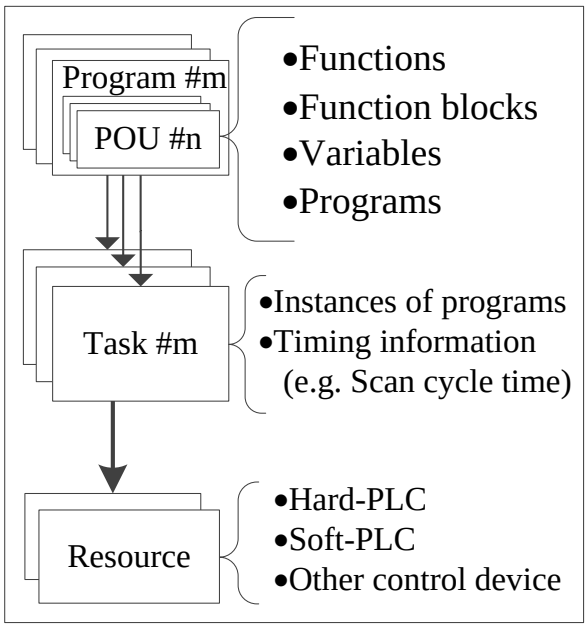
1. A novel MBD methodology for industrial the TTDRE systems and a complete tool-chain integrating off-the-shelf domain-specific tools and our developed interfaces & algorithms for automatically generating Soft-PLC programs.
   - Converting IEC 61131-3-based programs to intermediate representations using ANSI C for the purpose of estimating platform-specific timing information.
   - Estimating and back annotating the timing information of the control system to synthesize time accurate tasks and to reduce the design iterations.
   - Automatic synthesis of Simulink-generated FBs in IEC 61131-3 into the Soft-PLC-based TTDRE systems composed of tasks and a real-time scheduler.

2. A novel SILS framework where the physical plant is modeled in Simulink and the control program is executed in an open-source Soft-PLC (Beremiz [2, 172]) connected using CANOpen protocol [29].
   - Identifying various configuration knobs for design space exploration to meet the timing constraints during the SILS and to improve the QoC.

3.3 IEC-61131-3 Based Soft-PLC

PLCs (e.g. Siemens SIMATICS [3]) are proprietary cyclic TTDRE systems in the industrial control domain with various requirements such as hard real-time, extreme temperature environment and long life time [59]. Soft-PLCs or PC-based controllers (e.g. WinAC [4]) are soft real-time controllers that run on personal computers and may be used as a more affordable and flexible alternative of the traditional proprietary PLCs. PLCs/Soft-PLCs are

typically programmed using the IEC 61131-3 [31] language which is an independent standardized programming language for PLCs. Figure 3.2 shows the structure and an example of program using the IEC 61131-3 language. Details about PLC, Soft-PLC, and the IEC 61131-3 language may be found in [30, 31, 59]. In the scope of this paper, we are targeting Soft-PLC systems because Soft-PLCs are more generic and not vendor-specific, therefore our proposed methodology will be applicable to all types of Soft-PLCs. Moreover, Soft-PLCs are becoming a popular choice as controllers in the new era of industrial TTDRE systems (also know as industrial cyber-physical systems) for applications like building energy management, home automation, etc.



(a) *IEC 61131-3 language structure*

```
PROGRAM test_main
  VAR
    PLC_OUT : INT;
    PLC_IN : INT;
  END_VAR

  PLC_OUT := ADD(1, PLC_IN);
END_PROGRAM
```

(b) *ST program example*

```
CONFIGURATION conf

  RESOURCE res ON PLC
    TASK task0(INTERVAL := T#50ms,PRIORITY := 0);
    PROGRAM prog0 WITH task0 : test_main;
  END_RESOURCE

END_CONFIGURATION
```

(c) *Configuration example*

Figure 3.2: IEC 61131-3 language: (a) language structure; (b) and (c) examples of IEC 61131-3 languages

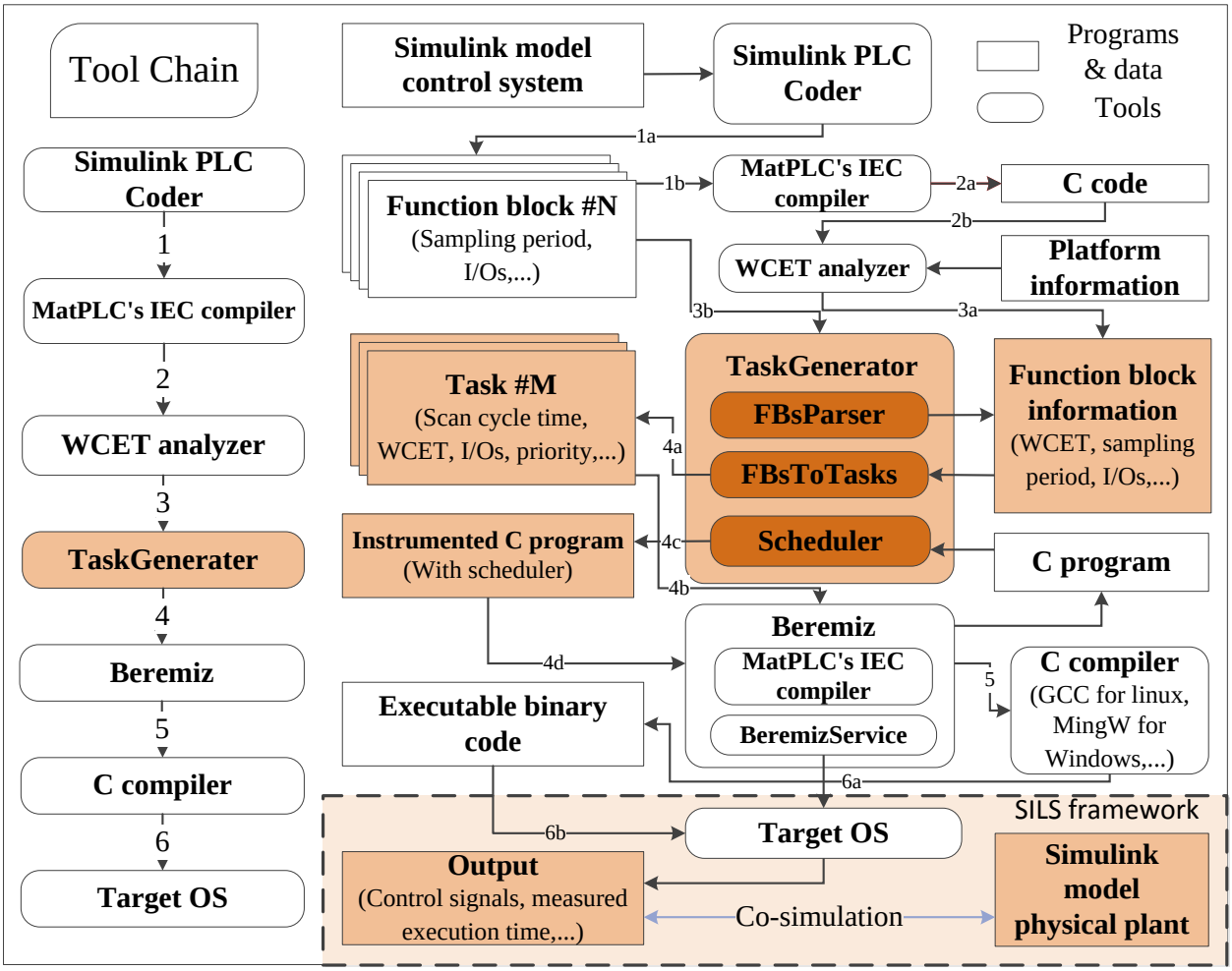


Figure 3.3: Our proposed MBD methodology and complete tool-chain (the highlighted parts are our contributions and rest of the parts are off-the-shelf tools/processes)

3.4 MBD of TTDRE for Soft-PLC Platform

3.4.1 Design Methodology

Various steps of our proposed MBD methodology and associated tools are presented in Figure 3.3.

Step 1: The first step of the proposed design methodology is to automatically generate non-executable FBs written in ST language of IEC 61131-3 standard using *Simulink PLC Coder* [5]. These FBs are used as the inputs for our proposed *TaskGenerator* tool that can

generate *Function Code* (FC)[3] and timing information of the tasks for the target Soft-PLC platform.

Step 2: In this step, we use an open-source IEC 61131-3 compiler [172] to translate the FBs into ANSI C-based FCs.

Step 3: FCs now may be used to analyze the timing information of the FBs based on the WCET of those generated FCs using the C language as an intermediate representation.

Step 4: During this step, *TaskGenerator* is used to generate the executable tasks with the timing information (see Section 3.4.2) and a real-time preemptive scheduler (see Section 3.4.3) for the run-time system. Moreover, timing and scheduling information is fed back to the system designer to achieve reduced iterations and shorter design-time to generate high fidelity system models for the subsequent stages.

Step 5: When tasks are automatically generated, this step implements the control program in a Soft-PLC.

Step 6: In this step, Soft-PLC programs are compiled for a target *Operating System* (OS) and made executable together with the provided Soft-PLC services.

3.4.2 Executable Task Generation

Our *FbsToTasks* (see Figure 3.3) tool generates a set of executable tasks from a set of non-executable FBs. The problem formulation for the tool development is given as follows:

Resource R is defined as: $R(PLM, CM, OS, CHC)$, where $R.PLM$ is the hardware platform, $R.CM$ is the compiler, $R.OS$ is the OS of the Soft-PLC, and $R.CHC$ is the communication channel capacity.

A function block FB is defined as: $FB(e, s, p, i)$, where $FB.e$ is the WCET on a particular resource R, $FB.s$ is the sampling period defined in the modeling tool, $FB.p$ is the priority of the FB, and $FB.i$ is the total bits of I/Os of the FB. Moreover, a set of n FBs is

[3]We use the term *Function Code* (FC) to refer to the code written in PC-centric programming language such as ANSI C generated from tasks written in IEC 61131-3 language.

defined as $\Phi = \{FB_1, FB_2, ..., FB_n\}$. A task τ is defined as: $\tau(e, s, d, p, i)$, where $\tau.e$ is the WCET of the task, $\tau.s$ is the *Scan Cycle Time* (SCT) [31] defined in the PLC program, $\tau.d$ is the deadline, $\tau.p$ is the priority, and $\tau.i$ is the total bits of I/Os of the task. In PLCs, the deadline of a task is the time of beginning the next scan cycle, so we can simplify the task as $\tau(e, s, p, i)$ by removing the deadline. Moreover, a set of m tasks T may be defined as $\{\tau_1, \tau_2, ..., \tau_m\}$.

Therefore, the problem solved in *FbsToTasks* tool is to find a mapping $\Phi \rightarrow T$ defined by a function $f(\Phi, R)$, where $m \leq n$. In this paper, we are only considering a Soft-PLC having a single resource R. Therefore, we divide the set of FBs Φ onto subsets $\varphi_1, \varphi_2, ..., \varphi_m$, such that inside each subset, all $FBs \in \varphi_j(s, n, i)$ have the same sampling rate. In a subset φ_j, $\varphi_j.s$ is the sampling rate of any $FB \in \varphi_j$, $\varphi_j.n$ is the total number of $FBs \in \varphi_j$, and $\varphi_j.i$ is the total bits of IOs for this FB subset. We denote a FB inside the subset φ_j as FB^j. Now, we create a set T of m tasks $\tau_1, \tau_2, ..., \tau_m$, map each task τ_j to φ_j by assigning $\tau_j.s$ to $\varphi_j.s$ and instantiate all $FB^j s$ inside τ_j. Finally, for all $FB^j s$, we count the total bits of non-duplicated I/Os and assign it to $\varphi_j.i$.

Intermediate Representation of IEC FBs

We follow a classical platform-based design approach [153] to generate executable tasks, so the capability of WCET analysis of the IEC 61131-3 based FBs needs to be achieved. For conventional PLCs, PLC vendors use proprietary hardware platforms ($R.PLM$) and PLC compilers ($R.CM$), therefore, the existing WCET analyzers which need R as an input may not be used. However, for a Soft-PLC, R can be easily obtained since the hardware is a standard computer. In our work, we have used the state-of-the-art WCET analyzer tool *AbsInt* [77] which takes R and a sequentially executed code segment as inputs to generate the WCET information for that code segment. However, existing WCET analysis methods are limited to inputs as ANSI C and/or executable binary code. Therefore, we translate the

set of tasks T in the IEC 61131-3 language onto ANSI C code using an IEC compiler [172].

Timing information Analysis of IEC FBs

After generating the WCET of all the FBs in Φ during the mapping stage for a task τ_i, the total execution time may be calculated as:

$$\forall FB^i \in \varphi_i : \tau_i.e = \sum_{1}^{\varphi_i.n} FB^i.e \tag{3.1}$$

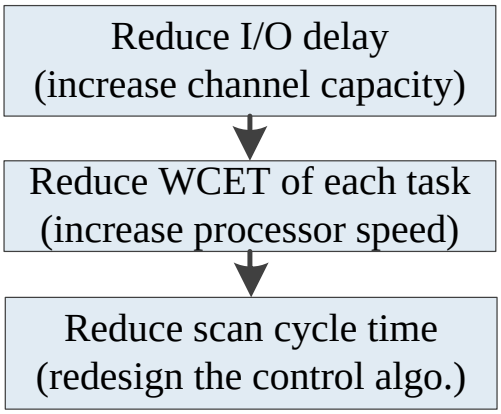


Figure 3.4: Presented configuration knobs

Besides the WCET, any possible communication delay that may occur in the final implementation should also be considered for reducing the design iteration of the industrial automation system. Therefore, for any task τ_i, with $\tau_j.i$ running on $R.CHC$, we estimate the communication delay as $(\tau_i.i)/(R.CHC)$. Finally, for any task τ_i, the processor utilization U_i may be calculated as:

$$U_i = ((\tau_i.e + (\tau_i.i)/(R.CHC)))/(\tau_i.s) \tag{3.2}$$

Based on the derived U_i of all the tasks, our *FbsToTasks* tool may report the schedulability of these tasks on the targeted resources. Therefore, the schedulability constraint will be:

$$\sum_{i=1}^{m} U_i \leq 100\% \tag{3.3}$$

Here, m is the total number of generated tasks. In summary, the *FbsToTasks* tool creates a report of the WCETs, the communication delays, and the result of the schedulability analysis of all the generated tasks and provides them back to the model designers to help reduce design

iterations. Considering Equations 3.1, 3.2, and 3.3, we present three configuration knobs for the model designer to reduce the processor utilization and increase the QoC of the system (see Figure 3.4): (1) to reduce the I/O delay; (2) to reduce the WCET of each task; and (3) to increase the SCT. With the help of our tool-chain, these three steps may be applied concurrently for reducing the number of design iterations.

Algorithm 1: Real-time preemptive scheduler for the executable tasks

Input: $T\{\tau_1, \tau_2, ..., \tau_n\}$: Original C code with a set of tasks
Input: *scp*: schedule cycle period
Input: *scn*: schedule cycle number
Input: THR_{rdy}: a list of threads ready to be executed
Output: $THR\{thr_1, thr_2, ..., thr_n\}$: Instrumented C code with a set of scheduled threads mapped from T

1 Set highest OS priority for current thread of scheduler; $THR = thr_1, thr_2, ..., thr_n; THR_{rdy} = \emptyset$;
foreach $thr \in THR$ **do**
2 $thr.P_{OS} = NORMAL$;
3 $thr_i.\tau_i = \tau_i$;
`/* Initialize all threads */`
4 $scp = GCD(\tau_1.s, \tau_2, s, ...\tau_n.s)$;
5 **while** *TRUE* **do**
  `/* Check and add new thread to be executed, reset the scn if necessary */`
6 **foreach** $thr_j \in ((scn \times scp)\%thr_j.\tau.s == 0)$ **do**
7 $THR_{rdy}.add(thr_j)$;
8 **if** $\tau_j.s$ *is the maximum of all* s *in* T **then**
9 $scn = 0$;
  `/* Update all threads in the list */`
10 **foreach** $thr_k \in THR_{rdy}$ **do**
11 $thr_k.P_{OS} = NORMAL$;
12 $thr_k.d = thr_k.\tau.s - (scn \times scp)\%thr_k.\tau.s$;
  `/* Prompt the thread/task with earliest deadline in list to HIGH priority */`
13 $\exists! thr_l \in THR_{rdy} : thr_l.d$ is the smallest in THR_{rdy};
14 $thr_l.P_{OS} = HIGH$;
  `/* Release the resource to run a cycle period */`
15 $scn++$;
16 Wait(scp);

Algorithm 2: Instrumentation of the Soft-PLC tasks

Input: A set of tasks
Output: A set of instrumented tasks
`/* Finish the running of task instructions */`
1 Execution of the current task
2 ; `/* Remove the current thread from the` THR_{rdy} `*/`
3 THR_{rdy}.remove(current thread); `/* Pickup a new thread to execute */`
4 $\exists! thr_l \in THR_{rdy} : thr_l.d$ is the smallest in THR_{rdy};
5 $thr_l.P_{OS} = HIGH$;

3.4.3 Automatic Synthesis of Real-time Preemptive Scheduler for the Executable Tasks

A real-time scheduler is the fundamental run-time system to guarantee hard real-time constraints for the generated executable tasks. However, an IEC 61131-3 language specific PLC-based TTDRE system does not have the required language semantics to support a scheduler for the generated tasks [31]. Typically, PLC vendors manage the scheduler manually and it is highly implementation specific and requires manual modification whenever there is a change in the system specification. Moreover, *Simulink PLC Coder* does not support generating any real-time scheduler for its non-executable FBs. To solve this problem, IEC-specific executable tasks are compiled to functions in C syntax and then necessary code instrumentation (see Algorithm 2) is performed to generate a real-time scheduler (see Algorithm 1) for the target Soft-PLC-based TTDRE system.

In our current work, *Earliest Deadline First Scheduler* (EDFS), which is a dynamic priority scheduling algorithm for a single core real-time system is automatically synthesized. We consider the following assumptions to generate our scheduler for the executable tasks : (1) $\forall \tau_i, \tau_j \in T, \tau_i, \tau_j$ are independent of each other; (2) $\forall \tau_i \in T$, the deadline is deterministic and equal to the sampling rate; (3) $\forall \tau_i \in T$, priority p is assigned dynamically based on the earliest-deadline-first conventions; (4) $\forall \tau_i, \tau_j \in T$, if $\tau_i.p > \tau_j.p$, then τ_i will be able to preempt τ_j immediately with a negligible preemption overhead.

Soft-PLCs (e.g. Beremiz) may utilize the standard thread services (which support pre-

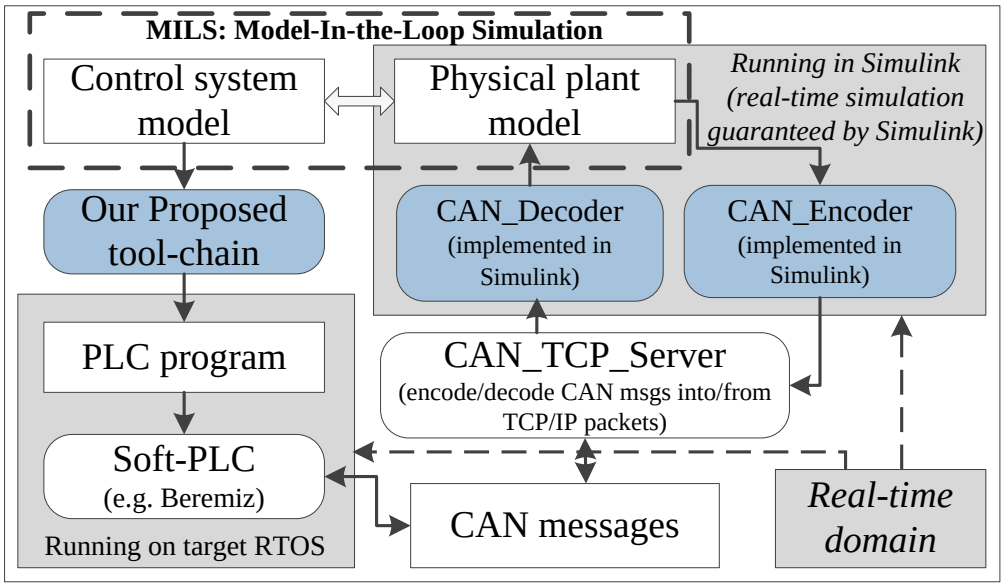


Figure 3.5: Our proposed SILS framework

emption capability required by the EDFS algorithm) provided by the host OS (e.g. Linux, Windows) to implement the run-time system for the EDFS algorithm. Therefore, during synthesis of the scheduling algorithm, we have implemented the EDFS by assigning each task function to a thread with the priority decided by its deadline. Algorithm 1 presents the pseudo-code of the automatically synthesized real-time preemption supported EDFS scheduler. In the algorithm, we define a thread as $thr(P_{OS}, \tau, d)$, where P_{OS} is the current priority of the thread in the host OS, τ is the task mapped to this thread, and d is the deadline of the current thread. We define the value of P_{OS}, to be either $NORMAL$ or $HIGH$. The *Greatest Common Divider* (GCD) of all the sampling period of tasks in T is assigned to *scp*. Either at the end of execution of a task (see Algorithm 2) or in every time stamp of *scp*, the scheduler will preempt the currently executing task, renew the list of tasks ready to be executed, and choose the task with the earliest deadline to be executed next by setting a $HIGH$ priority to it. The complexity of our synthesized algorithm is $O(n)$, where n is the number of tasks.

3.5 Validation Framework

3.5.1 Our Framework

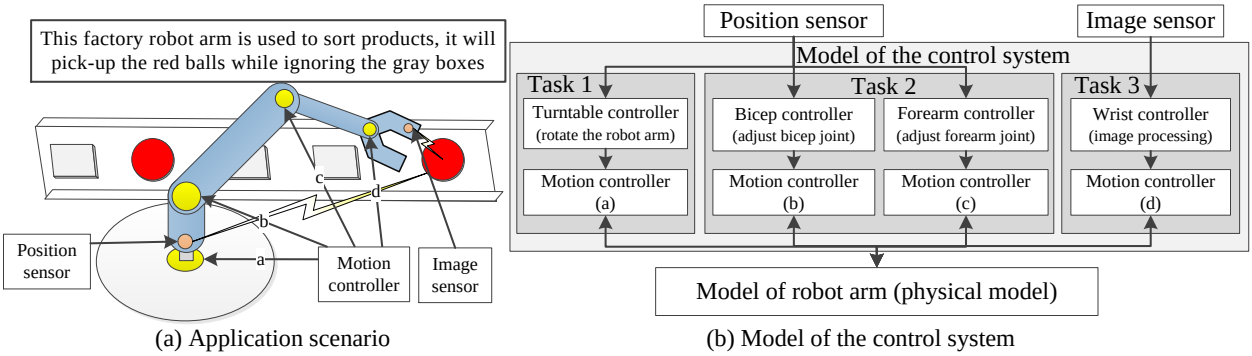


Figure 3.6: Application scenario of the factory robot-arm use-case

Traditional *Model-In-the-Loop Simulation* (MILS) provides only functional correctness and lacks any timing information of the control system and the physical plant. Therefore, industry has adapted the technique of HILS where real controllers and communication hardware are integrated in the control loop for accurate timing constraint validation. However, HILS requires a longer design cycle and is considered to be used during the end phase of the validation. Therefore, an emulation-based SILS approach where software-based control can be executed in a real control environment by keeping the plant and the communication as software models has been widely adopted [64]. In this paper, we present a SILS framework for our proposed MBD methodology of the Soft-PLC-based TTDRE systems (see Figure 3.5).

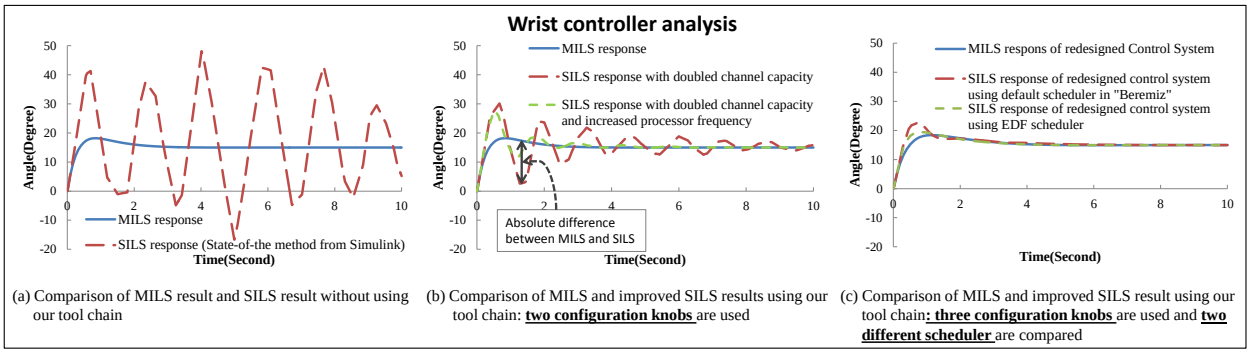
In our SILS framework, the physical plant is modeled using the physical modeling language (e.g. Simulink) and automatically generated, embedded control software (see Section 3.4) is executed on a Soft-PLC platform (e.g. Beremiz). To emulate the accurate communication-related timing constraints, the physical plant and the controller are connected through the TCP/IP channel with packets mimicking the messages according to the CANOpen protocol [29].

3.5.2 Communication and Synchronization

In our SILS framework, one CAN message (containing 64 bits of data) is encoded into one TCP/IP packet. From the plant side, TCP/IP packets may be received every block sample time defined in the TCP/IP blocks. Therefore, the channel capacity of our SILS framework may be calculated as:

$$ChannelCapacity = N_{message} \times 64bits/bst \tag{3.4}$$

Here, $N_{message}$ is the number of messages transferred per block sample time and bst is the block sample time of TCP/IP blocks. In this simulation framework, the synchronization problem is solved by running both the control program and the physical plant model in the same real-time domain. Figure 3.5 shows that the control program in the Soft-PLC uses the real-time service from the target *Real-Time Operating System* (RTOS) to guarantee the real-time execution and that the plant model (Simulink model in current work) uses the real-time synchronization tool-box.



(a) Comparison of MILS result and SILS result without using our tool chain

(b) Comparison of MILS and improved SILS results using our tool chain: **two configuration knobs** are used

(c) Comparison of MILS and improved SILS result using our tool chain**: three configuration knobs** are used and **two different scheduler** are compared

Figure 3.7: Wrist controller analysis using our tool-chain

3.6 Results AND Evaluation

For our experiment, we are using a real-world Soft-PLC based factory automation use case. Figure 3.6(a) shows the application scenario of the factory robot-arm application used to sort products. In this case study, the angles of the four robot-arm joints need to be controlled. The time-triggered control architecture of the application is shown in Figure 3.6(b). For our experimental purposes, embedded software is generated from the Simulink model using our tool-chain. Moreover, we have compared our experimental results with state-of-the-art methods/tools under two *Key Performance Indicators* (KPIs). Table 3.1 shows various control/platform-related parameters for the three tasks shown in Figure 3.6.

Table 3.1: Timing information of robot-arm control system

	Task 1	Task 2	Task 3	Processor utilization
WCET on initial processor	80ms	40ms	8ms	200.00%
I/O delay on initial channel	40ms	40ms	40ms	
Initial scan cycle time	200ms	100ms	80ms	
I/O delay on faster channel	32ms	32ms	32ms	178.00%
WCET on faster processor	20ms	10ms	2ms	110.50%
Redesigned scan cycle time	200ms	120ms	100ms	95.00%

The KPI we are using to compare the improvements of QoC using our tool is the *Sum of Absolute Differences* (SAD) between MILS and SILS results (see Figure 3.9). SAD is decreasing exponentially to 2.58% (indicating a 39X improvements of QoC) using our tools compared to the approach using *Simulink PLC Coder*. Figure 3.7(a) shows the MILS/SILS simulation results for the wrist controller without using our methodology and tool-chain. This graph presents the approach supported by Simulink for a target PLC. In this experiment we observe that there is a significant angular difference (degree) of the controlled robot-

arm joint between the MILS and the SILS outputs. Initially, the wrist controller has been designed with a sampling rate (SCT) of 80ms. Using our tool-chain, the timing information of *Task 3* is estimated to have a WCET of 8ms and I/O delay of 40ms. With these timings, the processor utilization (Equation 3.2) is calculated to be 200% as shown in Table 3.1. In order to achieve processor utilization that is lower than 100% (Equation 3.3) we have explored the three configuration knobs presented in section 3.4.2. To apply the first knob, we have decreased the block sample time of the physical plant (see Section 3.4.2) to reduce the I/O delay to 32ms. However, the processor utilization remains at 178% and therefore we have further applied the second configuration knob by increasing the processor frequency to achieve a processor utilization of 110.55% (see Figure 3.7(b)). Finally, we have used the third configuration knob to slightly increase the scan cycle time of *Task 2* and *Task 3* (Figure 3.7(c)). The result is shown in three steps; however our tool may use all three levels of optimization in a single design iteration. The processor utilization is reduced to 95.00% indicating a schedulable control program as shown in Table 3.1. Furthermore, Figure 3.7(c) compares the result of the wrist controller using the "Beremiz" scheduler [2, 172] and our implemented EDF scheduler. Figure 3.8 compares the number of deadline misses (KPI for scheduler in real-time system) using these two schedulers where our scheduler achieves the real-time requirement of zero deadlines misses compared to the Beremiz scheduler [172].

3.7 Conclusion

We have discussed our MBD methodology and associated tool-chain for designing the industrial TTDRE systems and a SILS framework for real-time validations of the generated control software. Our experiment shows a 39X improvement of QoC (2.58% SAD compared to [5]) with zero deadline misses that meets the hard real-time requirement (compared to [2, 172]) can be achieved using our tool-chain. We have also presented that the result shown

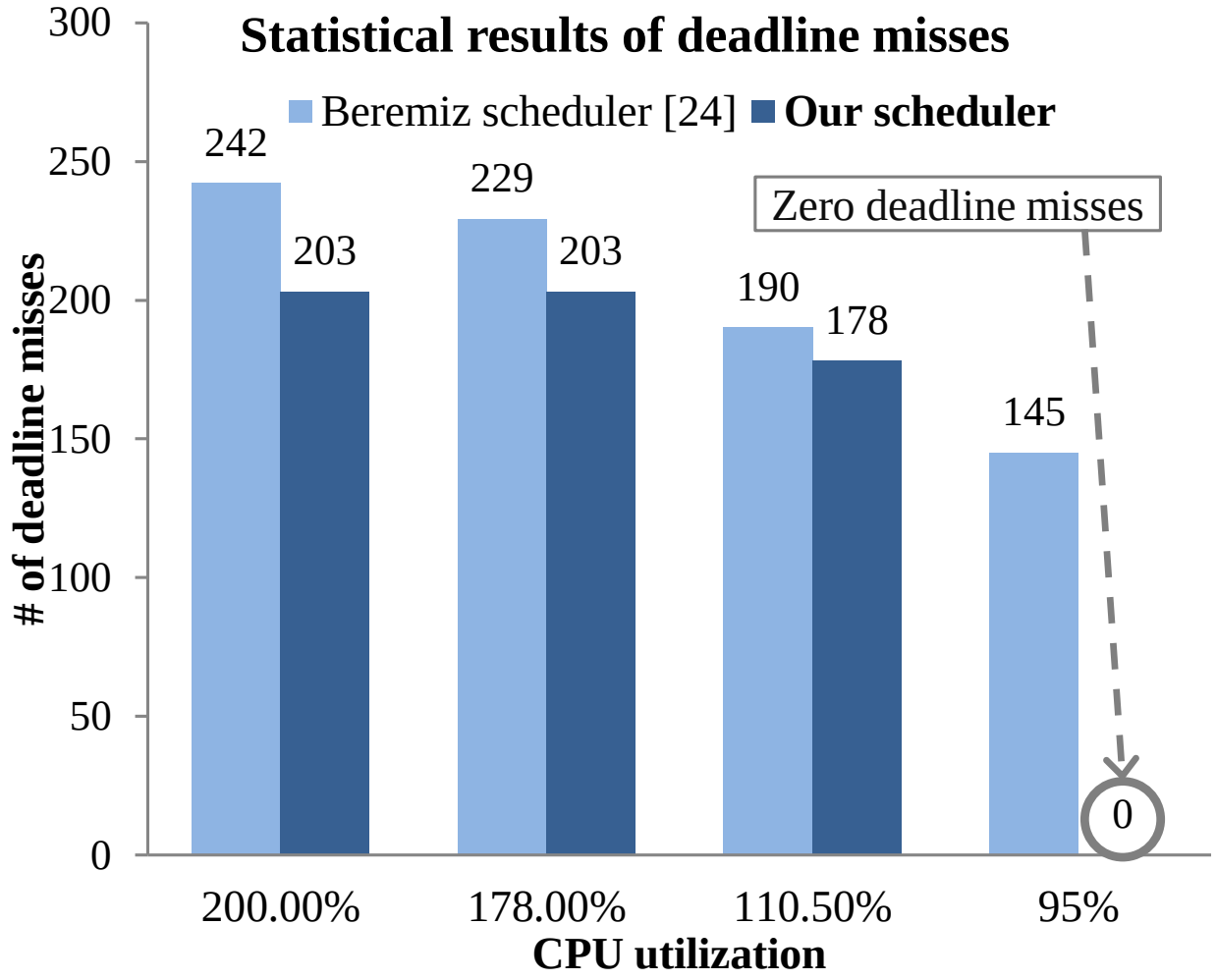


Figure 3.8: Comparison of our scheduler to Beremiz scheduler

in this case study may be obtained within a single design iteration while the traditional approach requires multiple iterations [28].

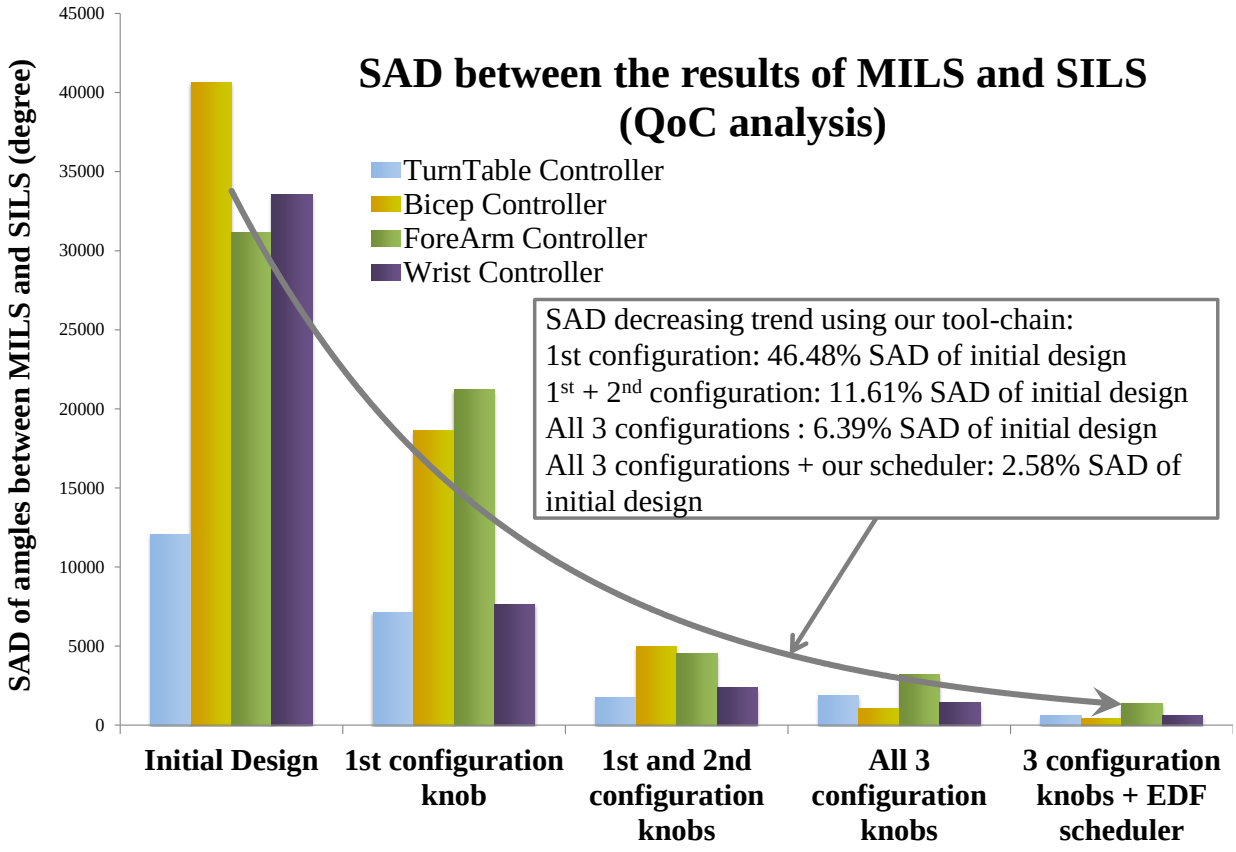


Figure 3.9: SAD between the results MILS and SILS

Chapter 4

Functional Model-based Design Methodology

4.1 Introduction

The high complexity of cross-domain engineering in combination with the pressure for system innovation, higher quality, time-to-market, and budget constraints make it imperative for engineers to use integrated engineering methods and tools. Computer engineering tools are mainly focused on a particular domain and therefore it is difficult to integrate different tools for system-level analysis. In this chapter, a novel multi-disciplinary systems engineering methodology and associated design automation algorithms for the complex automotive CPSs are presented. Rather than starting from the domain-specific architecture/simulation models where most resources are spent, we preemptively target the early design stage at the functional level that determines 75% of an automobiles cost. In our methodology, the marriage of systems engineering principles with high-level synthesis techniques from design automation area results in a functional modeling compiler capable of generating high-fidelity

simulation models for the design space exploration and validation of multiple cyber-physical automotive architectures. Using real-world automotive use-cases, we demonstrate how functional models capturing integrated cyber-physical aspects are synthesized into high-fidelity multi-domain simulation models.

Automotive Cyber-Physical Systems (CPSs) are some of the most technologically advanced and complex systems currently being produced. Modern automobiles are no longer mechanically-dominated systems; their physical behavior is greatly influenced by software (electronic control units [ECUs]) and network systems [11] – cyber components. For example, the remarkable advances in fuel economy, safety, performance, etc. have been possible due to the dense array of cooperating cyber components that interact with the multi-physics physical processes in an automobile. Designing a modern automobile with 100 ECUs [6, 39] controlling dozens of complex physical processes is a daunting task that requires the collaboration of hundreds of domain experts from various organizations. Thus, creating design automation tools that improve the automotive design process and allow companies to rapidly add new features, manage the heterogeneity of components, and maintaining the development time and cost targets is a challenging task [47].

State-of-the-art automotive design tools have been developed on the principle of *reducing the critical path* with the objective of making the design process shorter, cheaper, and more efficient. The critical path in a complex CPS design is the *detailed design* phase [7, 93, 173], where precise engineering specifications are created by the domain experts. Thus, mechanical engineering is supported by Computer-Aided Design (CAD) and Engineering (CAE) tools; electrical engineering by Electronic Design Automation (EDA) and wire harness design tools; control engineering by LabView [100], LMS [108], Modelica [123] and Matlab/Simulink [160]; and software engineering by UML [117] and in-house software development environments [25]. Unfortunately, because of the incompatibility between different domains, these tools cannot be easily and effectively combined to perform system-level analysis and simulations [53]. This

situation has led to a serious problem because the automobiles complexity keeps increasing but the design tools cannot answer simple but important questions such as estimating the impact of a change in the transmission architecture in the overall fuel economy of an automobile. Model Based Systems Engineering (MBSE) methodologies and tools [61] are becoming more popular as they allow ECUs to be designed, analyzed, implemented, and verified against system-level and high-fidelity multi-disciplinary systems in model-, software-, and hardware-in-the-loop simulations.

We take a novel approach for the creation of CPS design tools. We start with the *early design stage* which formalizes the requirements of a new automotive CPS in a *functional model* that defines *what* the system does in terms of energy, material, and signal flows. Functional models provide a high-level abstraction of the multi-disciplinary systems. It is used by practitioners to perform a broad Design Space Exploration (DSE) of various concepts and narrow down the design space to the few architectures that do satisfy the requirements. For example, the designer may quickly analyze the tradeoffs between various architectures for detecting humans using cameras, sonar, and LIDAR through simulation. Functional models are intimately related to *architecture*, defined as the allocation of physical structures to functionality [17, 24, 95]. From a design automation perspective, the high abstraction level in the functional models makes them a suitable formalism for CPS design. Functional models naturally express cyber-physical processes by abstracting details of the continuous and discrete dynamics, and naturally allow cross-domain collaboration [93, 173]. The novel contributions of this paper are as follows:

1. A practical approach using the Functional Basis language [81] (details in Section 4.2) as a higher-level abstraction for the formalization of the functionalities and for achieving the early DSE capability of the automotive CPS.

2. A Functional Model Compiler (FMC) that enables directly selecting architectures and generating high-fidelity multi-disciplinary simulation models from a functional model.

(a) A new "`Feedback`" function in the Functional Basis language [81] to facilitate the generation of closed loop system-level simulation models, an essential concept for control, software, sensors, and actuators of a CPS.

(b) A high-level mapping algorithm that translates functions to architecture components based on the contextualization of the components and (feedback) flows.

(c) A decision support system that uses a functional model *refinement process* to back-propagate the results of the high-level synthesis as the concept design suggestions.

3. A detailed case study of designing an automobile engine system that demonstrates the effectiveness of using our FMC to design the complex multi-domain and multi-disciplinary automotive CPS.

The rest of the paper is organized as follows. Section 4.2 introduces the background knowledge of our work and presents the related work. Section 4.3 presents the detailed implementation of the proposed FMC for synthesizing functional models into high-fidelity multi-domain simulation models. Furthermore, section 4.3.1 explains the proposed new "`Feedback`" function; Section 4.3.2 and section 4.3.3 present the detailed implementation of the proposed FMC in two separate steps, while the *contextualization-based mapping algorithm* is explained inside section 4.3.2; Section 4.3.4 presents our proposed *refinement process*. Finally, a detailed case study for automotive CPS application is discussed in Section 4.4 before concluding the paper in Section V.

4.2 Background and Related Work

During the early design stage of the complex systems, simulation is heavily used across industries because it is an economical and effective way to design, validate, and test. Systems

can be *virtually* analyzed without the need for physical prototypes because software is used to estimate the dynamic behavior of the system under a large variety of conditions. This, in principle, enables the early identification of system-level problems.

System simulation languages (e.g. VHDL-AMS [42], Simscape [112], Modelica [123]) allow the modeling and simulation of multi-disciplinary systems. Using a component-based approach, where behavior is specified according to energy conservation principles, these languages provide the means for multi-disciplinary modeling and simulation, where different engineering domains can be combined in the same language and in the same simulation. State-of-the-art automotive design tools [10, 108] start with these languages because they can unify multiple disciplines in the same language rather than using incompatible domain-specific tools. Unfortunately, obtaining a simulation model is currently a manual process that requires domain-experts from multi-disciplinary as a consequence of the lack of support for early stage design tools and methodologies. SysML [130] is proposed as a domain-independent system-level modeling language [9, 98, 186] providing a unified description of the functions and architectures of a system. However, SysML is limited to *express* models and does not specify any mechanism to *execute* these models [14].

Functional modeling is a systems engineering activity [57], where systems are described in terms of their functionalities and the functionalities of their subsystems. Fundamentally, a functional model reflects *what the system does*. A functional model decouples the design intentions (functions) from behavior and/or architecture, this improves the communication among different disciplines because it brings the minds of the domain experts to the same level of abstraction that is facilitated by natural language. In a functional model, specialized knowledge is irrelevant and discouraged. For example, the sentence "`transport people`" describes the goal of any transportation system and it intentionally ignores the different embodiments such as an automobile, a train, or an airplane. A functional model can also be refined into more specific descriptions in a process referred to as *functional decompo-*

sition. The "`transport people`" function can be decomposed into sub-functions such as "`navigate airspace`" or "`navigate highways`" implying the design of an airplane or an automobile. Furthermore, sub-functions can be decomposed creating a *functional decomposition tree* where the root node represents the top-level function and the leaf-nodes represent elementary functions such as "`convert chemical energy to mechanical energy`".

Formalization of functional model [81, 131] is critical for implementing this highly subjective and creative process into the systems engineering practice. An interesting practical problem of functional modeling is that different designers tend to create substantially different functional models of the same system. The Functional Basis language [81, 163] has been successfully implemented for function decompositions in the early design stage [27, 99, 146, 186]. By using a constrained vocabulary and well defined function and flow semantics, functional modeling practice may be normalized as different designers can rely on the same language to create and understand systems. As a result, it has become the de-facto standard for functional modeling and also to enable compatibility with other tools and methods [118]. Functional Basis defines three flow categories (*material*, *energy*, and *signal*) that expand into a total of 18 flow types, and 8 function categories with a total of 32 primitive functions.

Although Functional Basis language is successfully used as a multi-disciplinary communication language, the current major research challenge is to make it executable for performing extensive DSE. Therefore, generating an executable system-level models from the functional model to perform DSE is essential for the adoption of this methodology in industry. Existing work [78] has already discussed using functional model to synthesize architecture for automotive design, however, [78] did not provide a method to automatic the synthesis process. Moreover, [78] is not considering to utilize the functional model to help the design space exploration at the architecture level. In our preliminary work [35], we have demonstrated the possibility of using Functional Basis language to enable the early stage design automation for CPS. In this paper, by incorporating the decades of research in high-level synthesis for

electronic design [42, 175], we further apply the functional modeling technique to solve the multi-domain automotive CPS design problem at the early design stage.

4.3 Functional Model compiler (FMC)

The objective of the Functional Model Compiler (FMC) is to identify all feasible simulation models for a CPS concept, and down-select the ones that fulfill both the requirements (e.g. emissions) and the constraints (e.g. weight). Our FMC supports the automatic generation of automotive CPS simulation models (Definition 2) that are comparable in quality to manual selection of components done by experts from the functional models (Definition 1).

Definition 1. *A **functional model** is a labeled directed multi-graph $F = (V, E, s, t, L_V, L_E)$, where each node v_i is a function in this model, each edge $e_{(i,j)}$ represents the flow from v_i to v_j, $s : E \to V$ and $t : E \to V$ are two mapping indicating the source and target $v \in V$ of an $e \in E$. Each function $v \in V$ and each flow $e \in E$ have uniquely assigned labels $l(v) \in L_V$ and $l(e) \in L_E$ using the vocabulary from the Functional Basis language.*

Definition 2. *A **simulation model** S is a strongly typed component with well defined CPS ports $p_i \in P$ and connectors $c_{(p_i,p_j)} \in C$ that obey the energy conservation principles in various domains and allow components to exchange physical energy and data. Ports can be either physical or signal ports. Physical ports have an associated effort variable e – voltage, torque, pressure, temperature – and a flow variable f – current, angular velocity, volumetric flow, heat flow –, such that the system $e \cdot f$ represents physical power[1]. Signal ports are represented with numeric variables. Connectors enforce that only compatible port types are connected to each other.*

S encapsulates a description of hybrid behavior – continuous and discrete. Discrete behavior

[1]The Bond Graph foundations [37] allow the FMC to generate simulation models in various languages such as AMESim [108], Modelica [123], and Simscape [112].

is defined by a 5-tuple:

$$FSM = \{\Phi, \mathcal{I}, \mathcal{O}, \mathcal{U}, \Psi\} \tag{4.1}$$

where Φ is a finite set of states that may encapsulate continuous behavior, $\mathcal{I}$ is a set of input valuations, $\mathcal{O}$ is a set of output valuations, $\mathcal{U}$ is an update function that indicates a transition from a state and an input valuation to the next state and an output valuation, and Ψ is the initial state.

Continuous behavior is represented by a differential algebraic equation in the form of:

$$G(\dot{x}(t), x(t), z(t), u(t)) = 0 \tag{4.2}$$

where $x(t)$ is a vector of the time varying states variables of the system (e.g. velocity, temperature), $\dot{x}(t)$ the time varying state derivatives, z(t) the algebraic variables, and u(t) the inputs to the system.

Simulation parameters *$param_i(S)$ represent structural (i.e. component-dependent) and environmental (e.g. gravity) parameters. A* ***simulation library*** *$S_{lib} = (S_1, S_2, S_3, ...)$ is a collection of simulation components S_i.*

In our preliminary work [34, 35], we have established two basic approaches for implementing the FMC. In [34], we have developed a context-sensitive mapping algorithm to synthesize general purpose simulation models; however, when applied to more complex automotive applications, this approach leads to low fidelity simulation models because it does not leverage

any domain-specific knowledge. In [35], we have developed a high-level synthesis algorithm that utilizes the domain-specific knowledge in automotive architectures to generate simulation models; however, functions are directly mapped to library-specific components and this leads to the synthesis of medium fidelity models because all functionality is allocated to a single *viewpoint* of the problem that is provided by the mapped library. For example, the engine's functionality may be directly mapped to components in a drive-train sub-library that emphasize its mechanical architecture but ignores the electrical architecture.

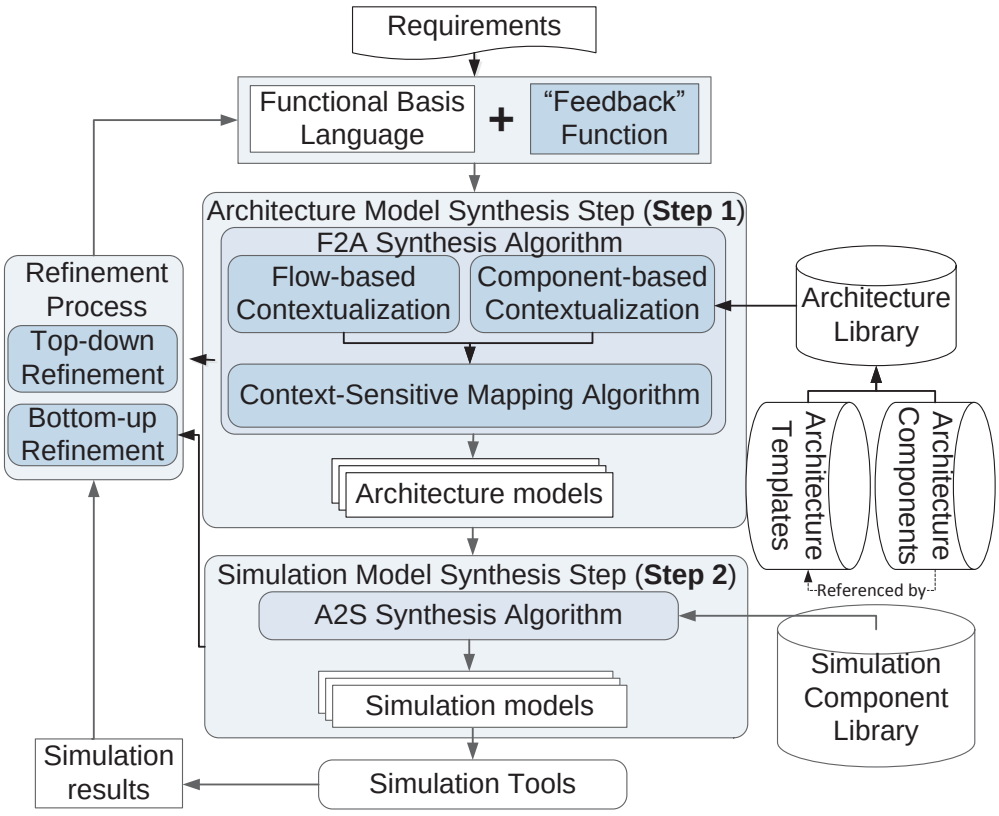


Figure 4.1: The overview of the FMC design flow.

This paper capitalizes on the key insights of these two approaches and develops a novel strategy for synthesizing high-fidelity multi-domain automotive CPS simulation models. The "Feedback" functions and architecture templates provide the basis for an *architecture-driven mapping of functions to components*. This mapping is further contextualized by the context-sensitive synthesis algorithm to generate simulations that take into account cross-cutting

concerns.

The overview of the FMC design flow is presented in Figure 4.1, our major contributions are highlighted and will be further explained in the following subsections.

4.3.1 Feedback Function

Most functional models lack feedback information because, in principle [131], a function is a process applied to an input flow to produce an output flow. Similarly, the entire system is seen as a black-box function that applies a process to its inputs to produce outputs. In our work, when the functional model is transformed into a system-level simulation model, the algorithm applies design rules to create feedback of flows and components with the help of a new proposed function type "`Feedback`" as a complementary to the Functional Basis language.

Definition 3. *The* ***`Feedback` function*** *has the following properties:*

1. *The syntax feeds back a flow produced by a successor function to a predecessor function (a function executed earlier in time relative to a successor)*[2].
2. *The semantics define the reuse or returning of material, energy, or signal flow to a predecessor function.*

Figure 4.2 shows the syntax and semantics of the proposed "`Feedback`" function. Currently, our method relies on general design and engineering rules created by the engineers to identify potential feedback loops in the functional models.

[2]The Functional Basis [81] specifies that functions execute chronologically from left to right.

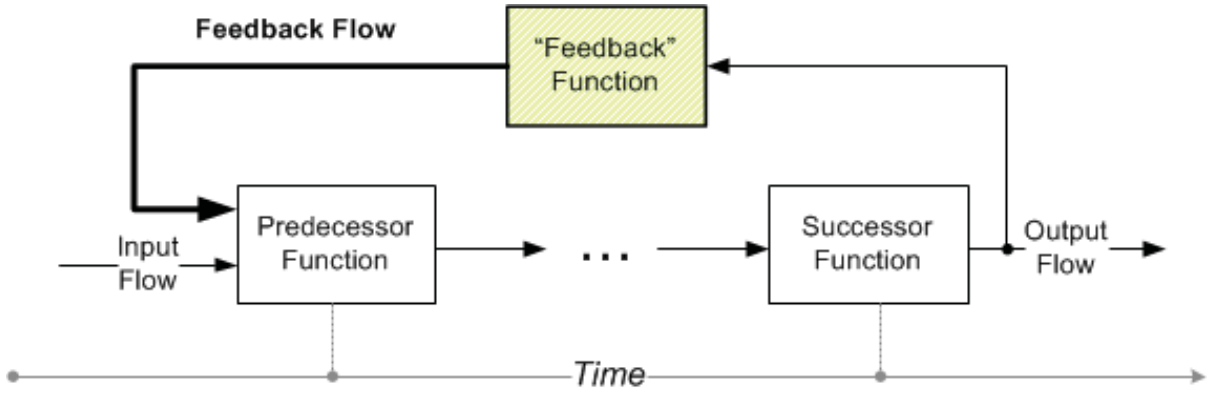


Figure 4.2: "`Feedback`" function semantics is the reuse or returning of material, energy, or signal flow to a predecessor function. Syntactically, it can be visualized as a flow from a `successor` to a `predecessor` function.

4.3.2 Architecture Model Synthesis Step (Step 1)

As presented in Figure 4.1, when the functional model is ready, the next step is to synthesize it into architecture models. Architecture-based design, also referred to as platform-based design [151] in the EDA community, allows companies to streamline the development process of complex systems across different organizations [24]. Architectures (or platforms) are estimated to save billions of dollars annually to companies because they allow re-usability of components across different models and brands [49]. To adopt the benefit from existing automotive architectures, we propose to use the existing knowledge of architectures to develop **Step 1** of the synthesis algorithm of our FMC. Our architecture synthesis algorithm supports the early design stage by synthesizing all the system-level potential architectural solutions that satisfy the design intent depicted by the designer in the functional model.

Definition 4. *An* ***architecture component*** *is a pair* $A = (F_{sub}, C_{list})$*, where* F_{sub} *is a functional model, and* C_{list} *is a list of constraints that specify relevant architectural parameters and properties such as the number of cylinders in an engine, data path width in an ECU, etc.*

Definition 5. *An* ***architecture template*** *is a multigraph* $ArchT = \{A, C, s, t, Cons\}$*, where* $a_i \in A$ *is an architecture component,* $c_{(a_i,a_j)} \in C$ *is the the connector connecting*

component a_i and a_j, $s : C \to A$ and $t : C \to A$ are two mappings indicating the source and target $a \in A$ of an individual $c \in C$, and $Cons = \{cons_1, cons_2, ...\}$ is a list of constraints that this architecture must meet.

Definition 6. *An **architecture library** $A_{lib} = (A_{set}, ArchT_{set})$ consists of two sub-libraries A_{set} and $ArchT_{set}$, where $A_{set} = (A_1, A_2, ..., A_n)$ is a collection of architecture components A_i, and $ArchT_{set} = (ArchT_1, ArchT_2, ..., ArchT_m)$ is a collection of architecture templates. The architecture components in $A_i \in A_{set}$ may be reused in the architecture templates $ArchT_{set}$ thanks to their object oriented properties.*

Definition 7. ***User given requirements** $R = \{req_1, req_2, ...\}$ is a set of requirements expressed in temporal logic that determine the expected system's characteristics. This representation is flexible to express different aspects of various cyber-physical domains and it admits a variety of efficient algorithms for translating a formula into verification code.*

The architecture synthesis algorithm uses the *Architecture Library* (Definition 6), composed by architecture components (Definition 4) and templates (Definition 5), to allocate functions to candidate architectures. Architecture libraries similar to ours are used in automotive design to document successful, unsuccessful, and reference designs. They also serve two additional purposes: (1) describe the structure and interfaces between reusable physical components, and (2) describe what functions are fulfilled by these. Our architecture components are object-oriented to facilitate re-usability.

Notice that the functional model created using the Functional Basis language allows the FMC to validate the design contracts [152] because the interfaces are strongly typed and can be mapped to CPS contexts (i.e. electrical, mechanical, signals, etc.) [34]. As a result, we propose a contextualization-based mapping algorithm for mapping functional models to candidate architectures in the following subsections.

Flow-based Contextualization

In order to reliably generate high-quality simulation models, every function within the functional model of a CPS must be put into context to be correctly mapped to its respective component(s). As previously stated, each function can be realized by an indefinite number of components, and each component can realize indefinite number of functions. To find the correct function-to-component mapping for a given model, each function must be contextualized by its input and output flows. We define two types of flows for every function: primary and secondary.

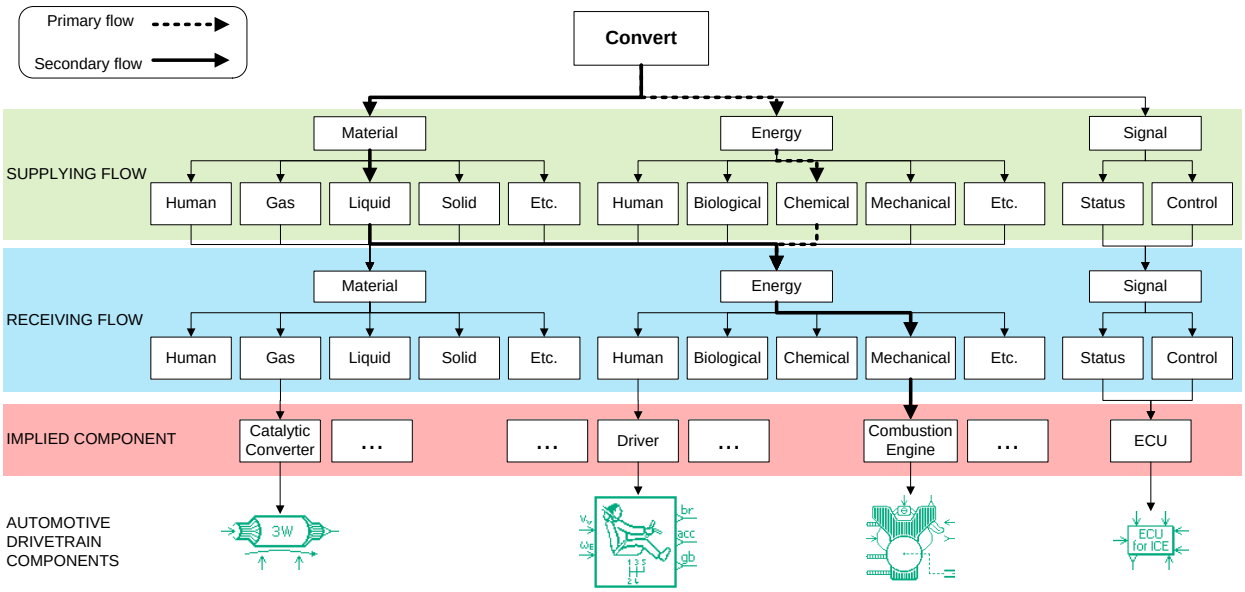


Figure 4.3: A context tree for the "Convert" function of the automotive system. The flows are separated into *supplying* and *receiving* flows. The supplying flow is linked with the input flow directly connected to the function, while the receiving flow is the output flow. By following the context tree, the algorithm is able to infer the correct components needed to realize the function. For example, following the highlighted flows, a combustion engine component is selected. Moreover, a catalytic converter may be selected if both the supplying and the receiving flows are gas; a human driver may be selected if the receiving flow indicates a human energy; ECUs may be selected if the flows are signals.

Definition 8. ***Primary flows*** *are the flows that are inherent to a given function (e.g. thermal energy flow in* `transmit thermal energy`*, or air flow in* `transport gas`*).*

As primary flows are fixed for every function, they add no new information to the system. Therefore, for flow-contextualization, we rely on *secondary* flows.

Definition 9. ***Secondary flows*** *are the non-essential inputs/outputs of a function (e.g. air flow in* `transmit thermal energy`*). Secondary flows help specify mechanisms that functions use, and possibly reduce the N-to-M function-to-component relation down to a 1-to-1 mapping.*

Components-based Contextualization

Reusing pre-existing system-level components is possible, thanks to the availability of academic and commercial libraries [102, 123, 124]. However, the components themselves are not sufficient for automatic synthesis (as we will explain in details in Section 4.3.4). It is very important to define the level of component granularity required by our algorithm to generate correct simulation models. For example, each simulation component (such as Modelica) defines both the structure (i.e. a capacitor) and its dynamic behavior using differential equations (Equation 4.2) and/or algorithms (e.g. $I(t) = C\frac{dV(t)}{dt}$). Additionally, a component's *connectors* (or ports) specify the equations to honor energy conservation principles. Finally, components have an *annotation* field that can be used to store information about the component such as its documentation or icon.

We use the name-space of the component in a library to classify its domain (e.g. *Modelica.Electrical.Analog.Basic*, *Modelica.Mechanics.Components*) and to locate the component (e.g. Resistor, Damper). Since our algorithm works at the component level, the equations and behaviors associated with the component are never modified. The type of the connectors in a component are used to determine the correct physical interface and generate compliant simulation models. Connector types are also useful to generate the energy conservation laws when a "`Feedback`" function relates various components. A component's annotation field

can be used as the means to associate and store the mappings of components-to-functions. Given the required level of component granularity, our algorithm imports the functions of a functional model and builds an abstract syntax tree to access a component's connectors, equations, algorithms, and annotations during mapping process.

Context-Sensitive Mapping Algorithm

In this section, we present our algorithm that enables context-sensitive mapping from functions to components. Before going to the detailed mapping algorithm, we first define a mapping from function to component as follows:

Definition 10. *A **mapping** from a function $v \in F$ to a component c is defined as a pairs $m = \{< v, c >\}$.*

The proposed mapping algorithm is presented in Algorithm 3. The required inputs are: a specific function, a functional model, and a set of potential components to be mapped. As defined in Definition 1, the functional model is a labeled directed graph $F = (V, E, s, t, L_V, L_E)$, where each function $v \in V$ and each flow $e \in E$ have uniquely assigned labels $l(v) \in L_V$ and $l(e) \in L_E$ using the vocabulary from the Functional Basis language and the newly introduced "`Feedback`" function (Definition 3) label. For example, a function w with $l(w) =$ `transmit` and an inbound flow f with $l(f) =$ `thermal energy` represent a `transmit thermal energy` function signature. Each component $c_i \in C_{in}$ is parsed into a query-able abstract syntax tree AST that, given a set of functions and flows $Q \subset F$, returns a list of components $C_{out} \in C_{in}$ that map to Q according to the function-component relations.

The mapping algorithm is presented in Algorithm 3. The The first step (Line 1) is to parse the input components into an Abstract Syntax Tree (AST) where equations, algorithms, connectors, and annotations are accessible for the algorithm. One context tree T is built for each function (Lines 3-6) while mapping the input function to the component (an example

Algorithm 3: Context-sensitive Mapping Algorithm

Input: Input function: v_{in}
Input: Functional model: $F = (V, E, s, t, L_V, L_E)$
Input: Input components set: C_{in}
Output: Mapping set: M_{out}

```
1  Build AST for all the component c_i ∈ C_in
2  M_out = ∅;
3  T.root = build signature from l(v_in) and primary inbound/outbound flows l(e_p) to/from v_in
4  T.inner = all possible combinations of secondary flows l(e_s) with v_in
5  CONTEXT = T.root × T.inner
6  T.leaf = deduce realization mechanism from CONTEXT
7  foreach levels i ∈ T do
8      create a path P_i that satisfies function v_in and all flows e connected to v_in
9  foreach c_i ∈ C_in do
10     if function-to-component and flow-to-component in AST matches for P_i then
11         M_out = M_out ∪ {< v_in, c_i >}
12 return M_out
```

of the context tree is show in Figure 4.3). Based on the context given by the function's signature (root node) and the secondary input/output flows (inner nodes), the realization mechanism is deduced from basic engineering principles and added as the leaf nodes. The context tree is then traversed (starting at the root) to create a path P_i according to the existing secondary flows and the appropriate realization mechanism (Lines 7-8). This path represents the flow-based contextualized function and all the functions and flows in P_i are mapped to the input components (Lines 9-11), thus creating a set of appropriate mapping M_{out}.

F2A Synthesis Algorithm

In this section, with the help of the proposed context-sensitive mapping algorithm, we introduce the high-level synthesis algorithm for synthesizing functional model to architectural models (F2A synthesis algorithm). The problem can be formulated as follows: given a functional model F and the user defined requirements R as inputs, the objective is to find the set of architecture templates $Arch_{mapped} \in A_{lib}$ that *fully* or *partially* map to F.

Definition 11. *A **mapping from a functional model** F **to an architecture template***

Arch is a set of mapping pairs $m = \{< v_i, A_j >\}$ *(Definition 10), where* $v_i \in V(F)$ *and* $A_j \in Arch$. *To represent all combinations, we define the mapping from* F *to Arch as* $MAP = \{m_1, m_2, ...\}$. *For convenience, we define* $A_j = \varnothing$ *if* v_i *does not exist. Furthermore, we define the mapping from* F *to all* $Arch \in A_{lib}$ *as a* $MAP_{set} = \{MAP_1, MAP_2, ...\}$.

Algorithm 4 generates both full and partial mappings MAP_{set} from a functional model F into a set of architecture templates $Arch_{mapped}$ based on the knowledge contained in the architecture library A_{lib}. Our synthesis algorithm also provides top-down $Arch_{refined}$ and bottom-up $F_{refined}$ refinements as design suggestions for F and $Arch_{mapped}$. Algorithm 4 performs a branch and bound synthesis [115] as follows. First, Lines 2-4 prune the selection of architecture templates whose constraints C_{list} do not meet the requirements R. Based on the selected architecture templates, Lines 5-11 aggregate the maximum possible mapping MAP_{set} from F to $Arch_{mapped}$ using the proposed Algorithm 3.

After the architecture template design space has been expanded, our algorithm performs a multi-step pruning process on each architecture template in $Arch_{mapped}$: (initial steps) Lines 12-16 eliminate from MAP_{set} the architecture templates whose architecture component constraints $C_{list} \in A_l$ violate any of the requirements R; (final steps) Lines 17-25 eliminate from MAP_{set} the architectures whose connectivity do not match the functional connectivity (flows) in F; and eliminates the architecture templates whose architecture components were eliminated entirely by the multi-step pruning process.

Lines 26-29 generate a top-down refinement $A_{refined}$ for all partial mappings; and Lines 30-33 generate the bottom-up refinement $F_{refined}$ based on the functions F_{sub} that exist on the selected $Arch_{mapped}$ but not in F. The refinement suggestions can help the designers to increase the fidelity of the original functional model F. The details of refinement process are presented in Section 4.3.4. Finally, all the generated mappings and refinement suggestions are returned at Line 34.

Algorithm 4: F2A Synthesis Algorithm

Input: F: A functional model
Input: R: User defined requirements
Input: A_{lib}: An architecture library
Output: MAP_{set}: A set of mappings from F to $Arch_{set} \in A_{lib}$
Output: $Arch_{mapped}$: A set of mapped architecture templates
Output: $Arch_{refined}$: A set of refined architecture templates
Output: $F_{refined}$: A set of refined functional models

$Arch_{mapped} = Arch_{set} \in A_{lib}$; $MAP_{set} = \varnothing$
foreach $Arch_i \in Arch_{mapped}$ **do**
  if $\exists Cons_j(Arch_i)$ *contradicts with any* $req_k \in R$ **then**
    $Arch_{mapped} = Arch_{mapped} \setminus Arch_i$
foreach $Arch_i \in Arch_{mapped}$ **do**
  $MAP_i = \varnothing$
  foreach $v_j \in F$ **do**
    $MAP_i = MAP_i \times \{< v_j, \varnothing >\}$
    A_{set} = Find all $A_k \in Arch_i$
    MAP_i = Context-SensitiveMapping(F, v_j, A_{set})
  $MAP_{set} = MAP_{set} \cup MAP_i$
foreach $MAP_i \in MAP_{set}$ **do**
  foreach $m_j \in MAP_i$ **do**
    foreach $< v_k, A_l > \in m_j$ **do**
      if $\exists C_{list}(A_l)$ *contradicts with any* $req_m \in R$ **then**
        $MAP_i = MAP_i \setminus m_j$
  foreach $m_j \in MAP_i$ **do**
    foreach $e_k \in E(F)$ **do**
      $v_s = s(e_k) \in V(F)$;$v_t = t(e_k) \in V(F)$
      Find A_t in $< v_t, A_t > \in m_j$ and A_s in $< v_s, A_s > \in m_j$
      if $A_s \neq A_t$ *and* $\nexists c_{(s,t)} \in C(Arch_i)$ **then**
        $MAP_i = MAP_i \setminus m_j$
  if *All mappings in* $MAP_i =< v_k, \varnothing >$ **then**
    $Arch_{mapped} = Arch_{mapped} \setminus Arch_i$
    $MAP_{set} = MAP_{set} \setminus MAP_i$
  foreach $m_j \in MAP_i$ **do**
    foreach $< v_k, A_l > \in m_j$ **do**
      if $A_l = \varnothing$ **then**
        $Arch_{refined} = Arch_{refined} \cup \{Arch_i \cup v_k\}$
foreach $Arch_i \in Arch_{mapped}$ **do**
  foreach $\exists f_{subj} \in F_{sub}(A \in Arch_i)$ **do**
    if $\exists f_{subj} \notin F$ **then**
      $F_{refined} = F_{refined} \cup \{F \cup f_{subj}\}$
return MAP_{set}, $Arch_{mapped}$, $Arch_{refined}$ and $F_{refined}$

Let the total architecture components in the architecture library $A_i \in A_{lib}$ be N, total architecture templates in the architecture library $Arch_i \in A_{lib}$ be M, and the number of

functions in F to be K. The aggregation step in Lines 5-11 has a complexity of $\mathcal{O}(N*M*K)$. The multi-step pruning process in Lines 12-25 is a search of all the possible mappings and each step has a complexity of $\mathcal{O}(N*M*K)$. Thus, Algorithm 4 has a complexity of $\mathcal{O}(N*M*K)$.

4.3.3 Simulation Model Synthesis Step (Step 2)

The **Step 2** in FMC is to generate the corresponding simulation model for each of the candidate architectures identified by Algorithm 4 in order to analyze the actual performance against the expected performance of the system and to conduct trade-off analysis between the different candidate architectures.

We use a novel automated approach for extracting functionality from simulation components. We perform an automatic interface analysis on port types, a structural (i.e. equational and algorithmic described using Equation 4.2 and Equation 4.1) analysis on the mathematical model, and a parameter analysis on the component's parameters to determine what functionalities it fulfills. Note that the functional extraction is performed once per library and the identified functions are automatically annotated in the library as additional semantic information for future reuse. Although, we performed functional extraction on the AMESim library [108], this technique is applicable to any other physical modeling language that uses effort-flow variables and differential equations, such as Simscape [112] and Modelica [123].

Definition 12. *A **mapping from an architecture-mapped functional model to simulation components** is a set of 3-tuples $t = \{< v_i, A_j, S_k >\}$, where $v_i \in V(F)$, $A_j \in Arch$, and $S_k \in S_{lib}$.*

Algorithm 5 generates simulation models for all the architecture-mapped functional models (A2S synthesis algorithm) identified by Algorithm 4 in three sub-steps. Sub-step 1 (Lines 2-15) broadens the simulation design space $S_{solution}$ for each of the architecture templates

Algorithm 5: A2S Synthesis Algorithm

Input: F: A functional model
Input: R: User defined requirements
Input: S_{lib}: A pre-computed simulation library with semantic annotations
Input: A_{lib}: An architecture library
Input: MAP_{set}: A set of mappings from F to $Arch_{set} \in A_{lib}$
Output: $F_{refined}$: A set of refined functional models
Output: $Arch_{refined}$: A set of refined architecture templates
Output: $S_{refined}$: A set of refined simulation components
Output: S_{set}: A set of simulation models

1 $S_{set} = S_{refined} = Arch_{refined} = F_{refined} = \varnothing$
2 **foreach** $MAP_i \in MAP_{set}$ **do**
3 $S_{solution} = \varnothing$
4 **foreach** $m_j \in MAP_i$ **do**
5 **foreach** $< v_k, A_l > \in m_j$ **do**
6 $S_{interface}$ = find $P(A_l)$ in $P(S_{lib})$
7 $S_{equations}$ = find $P(v_k)$ in $P(S_{lib})$
8 $S_{parameter}$ = find $C_{list}(A_l)$ in $param(S_{lib})$
9 $S_{functional}$ = find $l(v_k), l(s(v_k)), l(t(v_k))$ in $\sigma(S_{lib})$
10 $S_{matches} = S_{interface} \cap S_{equations} \cap S_{parameter} \cap S_{functional}$
11 **if** $S_{matches} = \varnothing$ **then**
12 $S_{refined} = S_{refined} \cup < v_k, A_l, \varnothing >$
13 **else**
14 **foreach** $S_m \in S_{matches}$ **do**
15 $S_{solution} = S_{solution} \cup < v_k, A_l, S_m >$
16 **foreach** $s_k \in S_{solution}$ **do**
17 $S_{set} = S_{set} \times s_k$
18 **foreach** $S_i \in S_{set}$ **do**
19 $\forall c(i,j) \in C(Arch(S_i)) \rightarrow c(i,j) \in C(S_i)$
20 **foreach** $P(S_i) = \varnothing$ **do**
21 $S_{sources}$ = find $P(S_i)$ in $P(S_{lib})$
22 S_{sinks} = find $P(S_i)$ in $P(S_{lib})$
23 $P(S_i) = P(S_i) \cup S_{sources} \cup S_{sinks}$
24 $P(S_i) \rightarrow P(Arch_{refined}(S_i))$
25 $P(S_i) \rightarrow P(F_{refined}(S_i))$
26 **return** S_{set}, $S_{refined}$, $Arch_{refined}$ and $F_{refined}$

$A_l \in MAP_{set}$. The temporary variable $S_{matches}$ is a pruned set of simulation components from S_{lib} that fulfills the functionality specified in $< v_k, A_l >$. Lines 6-10 perform a search on the simulation library S_{lib} to identify the most appropriate components that match the interface $S_{interface}$, energy transformations $S_{equations}$, parameters $S_{parameter}$, and the manually annotated functional information $S_{functional}$. The matched simulation components $S_{matches}$ are then used to create the 3-tuples that define the mappings (Definition 12) of architecture-mapped functional models to simulation components, and are added to the simulation design

space $S_{solution}$. Whenever an architecture-mapped function cannot be mapped to any simulation component, we construct a top-down refinement of simulation models $S_{refined}$ (See details of refinement process in Section 4.3.4). Sub-step 2 (Lines 16-17) expands the simulation design space S_{set} by creating individual simulation models for all possible combinations of simulation components that match an architecture-mapped functional model. Sub-step 3 (Lines 18-25) adds connections to every model in S_{set} according to the topology in their corresponding architecture component. Ports $P(S_i)$ without connections are connected to the appropriate sources and sinks simulation components (e.g. constants, terminals, grounds, etc.); these ports and their new connections to sources and sinks are used to generate the bottom-up refinement in $Arch_{refined}$ and $F_{refined}$.

Let the candidate architecture templates $Arch_{set} \in MAP_{set}$ be N, and the number of simulation components in S_{lib} be M. Finding matches from architecture components $A_l \in Arch_{set}$ to simulation components $S_{matches}$ – Lines 4-15 – has a linearithmic time complexity $\mathcal{O}(NlogM)$ because the simulation library has been pre-processed and the functional information has been already extracted and organized for efficient search; pre-processing has a linear time complexity $\mathcal{O}(M)$. The topology construction in Lines 18-25 has a time complexity proportional to $\mathcal{O}(C(Arch_{set})*S_{set})$. In practice, $C(Arch_{set}) \approx N$ and $M >> S_{set}$; therefore, the time complexity of Algorithm 5 is $\mathcal{O}(NlogM)$. The simulation design space S_{set} created by Lines 16-17 has a space complexity of $\mathcal{O}(N*N_{A_l}*M_{S_{matches}})$.

4.3.4 Refinement Process

Automotive design is a complex iterative process; one design is passed hundreds of times through hundreds of personnel in various organizations. In most cases, the first functional model is a low fidelity model that captures the essence of the design and defines only the basic set of functions F_A to fulfill the high-level requirements. Our synthesis algorithm supports

the refinement of such low fidelity functional models by leveraging the company's know-how that exists in the architecture library. For example, after the initial candidate architectures have been identified, the architecture models may contain a set of functions F_B that are not modeled in the original functional model. Thus, this information can be propagated back to create a refined functional model $F_C = F_A \cup F_B$. We refer to this refinement as *bottom-up refinement* because information is being propagated from a lower level of abstraction (architecture) to a higher-level of abstraction (functions) to make it less incomplete.

Definition 13. *The* ***level of fidelity*** *is the amount of qualitative and quantitative information that can be obtained from a model.*

Definition 14. ***Incompleteness*** *of a model is the lack of fidelity necessary to answer a specific engineering question.*

Definition 15. ***Refinement*** *is the process by which the level of fidelity of a model is increased to make it less incomplete, and thus the ability to answer more detailed engineering questions.*

Top-down Refinement

A *top-down refinement* propagates information from a higher level of abstraction (functions) to a lower level of abstraction (architecture) to increase the level of fidelity. For example, when a functional model F contains a set of functions that are not fulfilled by any of the components in the architecture library A_{lib}, then a new architecture component A_{new} with the relative complement of the functions in the architecture library in the functions in the functional model is created by $A_{new} = F \backslash A_{lib}$. The new architecture component A_{new} is then subject to having a design contract signed-off by an engineer to describe how it relates vertically to other levels of abstraction, horizontally to other architecture components, parameterized according to architectural parameters of interest, and uniquely named and

described. Although these suggestions are provided to the designers, our algorithm only uses architecture components that have been signed-off by an engineer and have a design contract in place. This guarantees that the high-level synthesis is performed with the validated models.

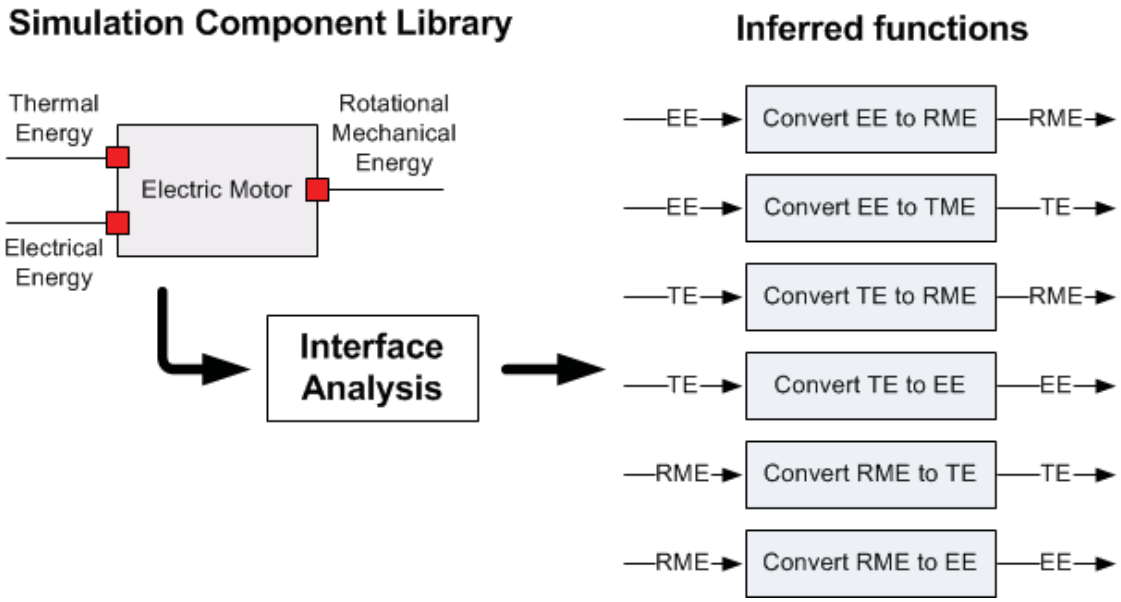


Figure 4.4: Extraction of functions from simulation components.

Bottom-up Refinement

Our *bottom-up refinement* automatically determine the set of functions achieved by a simulation component through analyzing its ports and interface, as well as determining the flow (energy, material, signals) transformations that occur within the component's internal structure. This bottom-up step classifies components in a library of simulation components S_{lib} according to the functions they perform. This classification information is important in order to achieve a correlation between functions, architectures, and simulation components that design tools can use to synthesize simulation models for candidate system architectures.

Analyzing the components' interface determines what type of energy, material, or signals can be exchanged with other components through its ports. Figure 4.4 shows how the

interface analysis infers the functions achieved by a given simulation component. For example, the `Electric Motor` component has three ports: thermal, electrical, and rotational mechanical energy. Typically, simulation components exchange energy through a pair of physical variables known as conjugate variables that represent effort/flow. The conjugate variables voltage/current represent electrical energy, torque/angular velocity represent rotational mechanical energy, and temperature/heat flow represent thermal energy. Because the relationship between energy at the functional level, and energy at the simulation component level is known and well defined, our algorithm determines that the `Electric Motor` is able to perform the function of "`Convert`" energy from any port to the other ports. Thus, a total of six functions are inferred.

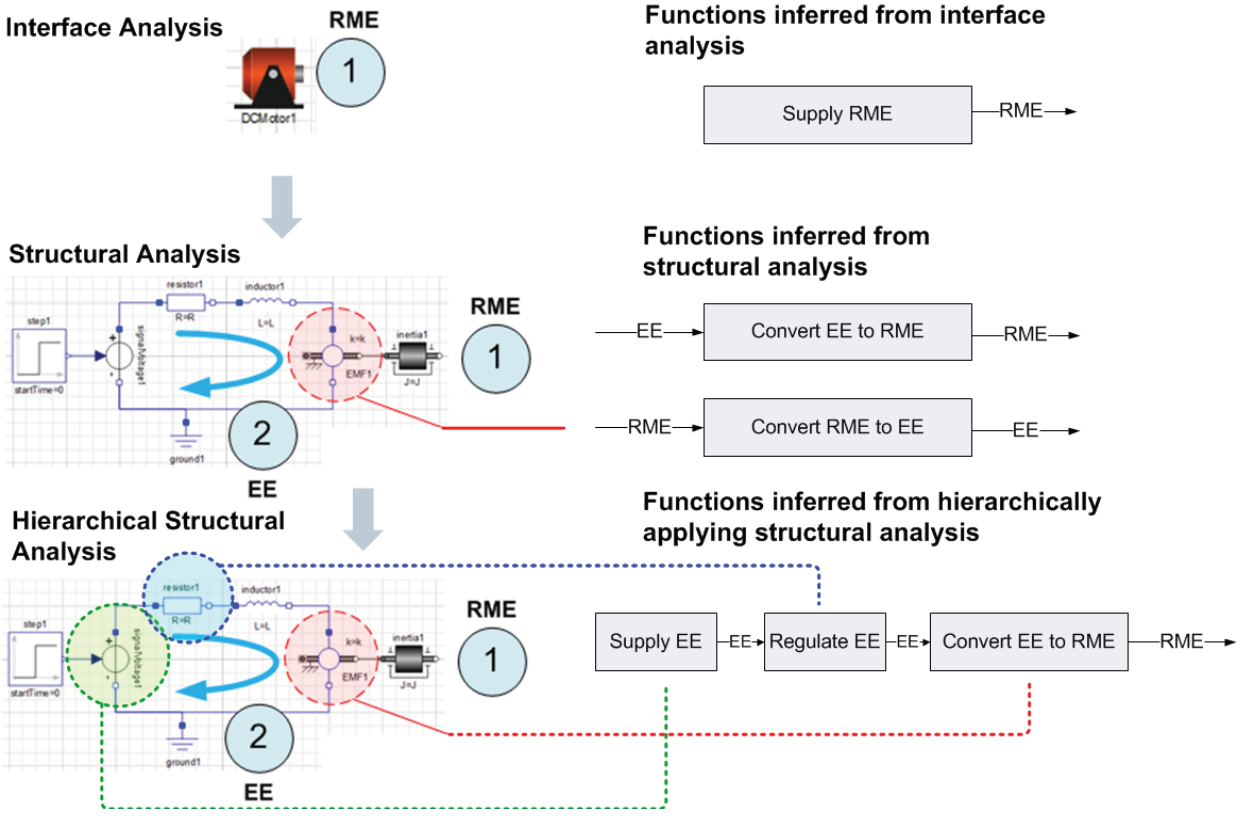
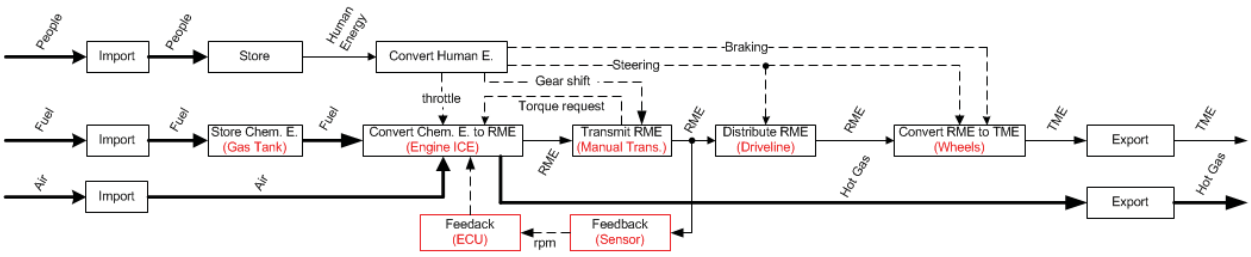


Figure 4.5: Structural analysis on simulation components reveals additional domains and the energy, material, signal transformations between them.

In some cases, simulation components are modeled with additional domains *that are not visible through their interface* and therefore our algorithm performs a structural analysis



(a) Functional model of an automotive.

ID	Description	Value
$r1$	Minimum energy effiency	35%
$r2$	Maximum weight	2000KG
$r3$	Maximum price	$40,000

(b) User defined requirements R .

ID	Description	ICE	Series Hybrid	Parallel Hybrid	Electric	Electric Fuel Cell
c_1	Max. Energy effiency	20%	37%	37%	60%	50%
c_2	Min. Total weight	2000 kg	2000 kg	2000 kg	2000 kg	2000 kg
c_3	Min. Price	$20,000	$30,000	$30,000	$30,000	$50,000
c_4	Input Energy	Chemical	Chem., Electric	Chem., Electric	Electric	Electric
c_5	Output Energy	Mechanical	Mechanical	Mechanical	Mechanical	Mechanical
c_6	Energy Price	$3 per gallon	$0.23 per kWh	$0.23 per kWh	$0.23 per kWh	$0.15 per kWh

(c) Constraint list C_{list} of architecture templates $Arch$.

Figure 4.6: An example functional model of an automotive power-train, associated requirements, and architectural constraints.

– a hierarchical traversal and flattening – of the internal simulation component structure to expose all the domains (e.g. electrical, mechanical, thermal, signals, etc.). Figure 4.5 shows the interface of the `DCMotor1` component of the Modelica Standard Library with a single port (Rotational Mechanical Energy or RME). The interface analysis infers a single function "`Supply RME`". However, the structural analysis exposes the internal composition of the `DCMotor1` component in terms of two domains: `RME` and `EE`. Notice that although electrical components exist, they are not exposed in the component interface and performing an interface analysis would only lead to a correct, but not very detailed functionality of the simulation component. Therefore, using the structural analysis our algorithm identifies two additional functions "`Convert EE to RME`" and "`Convert RME to EE`".

A further level of structural analysis, referred to as hierarchical structural analysis, is shown in Figure 4.5. By applying the same principle to the electrical components in the `DCMotor1`, a graph of functions can be inferred by the algorithm. For example, the `signalVoltage1` is connected to `Resistor1`, and this is connected to `EMF1` simulation components. Using the same structural and interface analysis on the individual components, our algorithm determines the flow of electrical energy as a graph of functions where "`Supply EE`", "`Regulate EE`", and "`Convert EE to RME`" functions are connected. This provides a more detailed functional understanding of a component that can be used to drive the synthesis of architecture-

mapped functional models according to different levels of fidelity.

4.4 Case Study

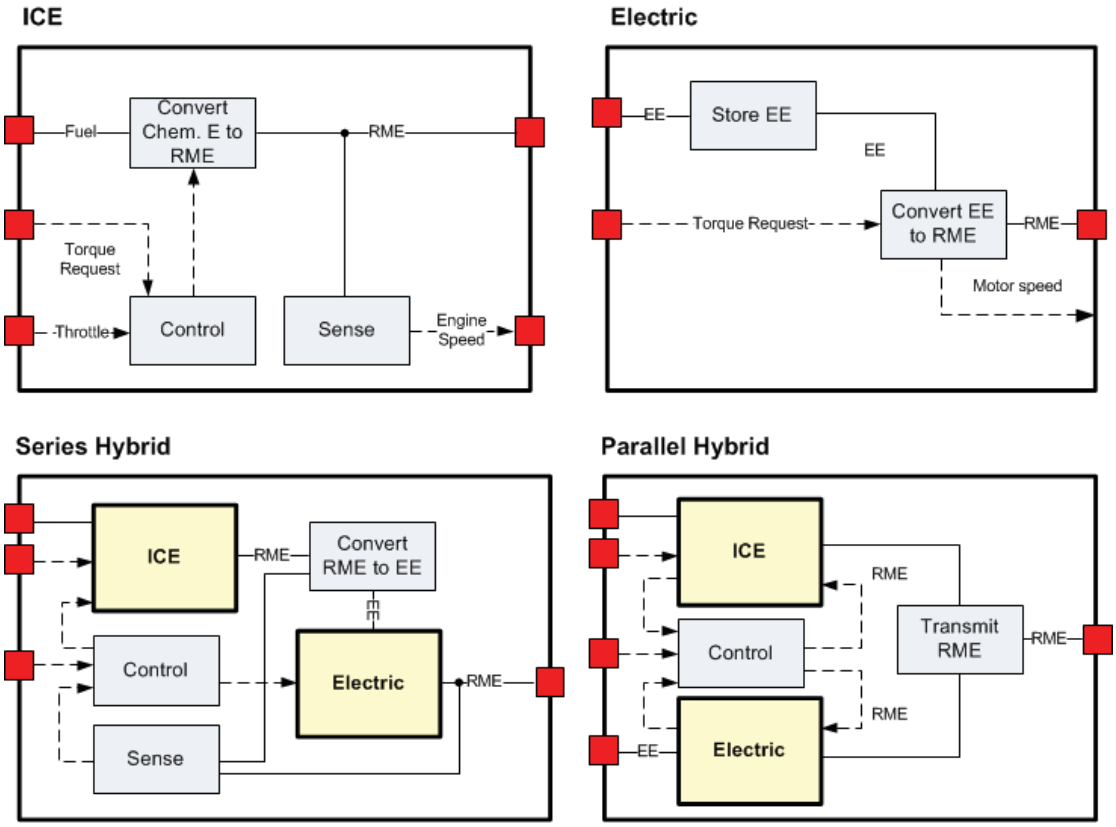


Figure 4.7: Architecture templates for the engine block.

In this section, we use a real-world automotive case study to demonstrate how our FMC may capture the cyber-physical aspects of a functional model and synthesize it into multi-domain simulation models. We evaluate several different architectures of designing an engine system using our proposed methodology capturing the knowledge of existing architectures and generating the simulation models automatically. We furthermore demonstrated the effectiveness of having the proposed "`Feedback`" function and the refinement process for generating the high fidelity multi-domain simulation models.

4.4.1 Synthesis of Architecture Models

A functional model of the engine system is shown in Figure 4.6. Notice that the functions (blocks) and flows (arrows) can be naturally mapped to the main subsystems of an automotive represented by architectural templates shown in Figure 4.7. For example, the function "`convert chem. energy to rot. mech. energy`" function can be mapped to an `Engine ICE`, `Series Hybrid`, and `Parallel Hybrid` architectures. In this section, we focus on the design of the engine system.

The user defined requirements R are given in Figure 4.6(b), and the constraints C_{list} for five automotive architecture templates are given in Figure 4.6(c). Notice that the $c3$(`Electric Fuel Cell`) violates $r3$ and therefore this is eliminated from the architectural design space by Algorithm 4. Furthermore, Algorithm 4 also eliminates the `Electric` architecture template because neither "`Store EE`" nor "`Convert EE to RME`" functions exist in the functional model. Notice that while the `ICE` architecture template fully matches the functional model, the `Series Hybrid` and `Parallel Hybrid` templates *partially* match the functional model F.

4.4.2 Synthesis of Simulation Models

Using functional models, the design intent remains technology-independent and the FMC allows non-experts to automatically generate simulation models for various architectures that satisfy the design requirements. For example, the functional model in Figure 4.6 is created by a software engineer with marginal understanding of mechanical engineering principles who wanted to design the engine control system (ECU and its control software) represented by the "`Sense`" and the "`Control`" functions. Figure 4.8 shows the AMESim simulation model for the `Series Hybrid` architecture synthesized by the FMC. Notice that our high-level synthesis algorithm appropriately down-selects the `VCU-SH` ECU simulation component to

control both the `ICE Engine` and the `Electric` motor as per the topology and port structure defined in the `Series Hybrid` architecture template in Figure 4.7.

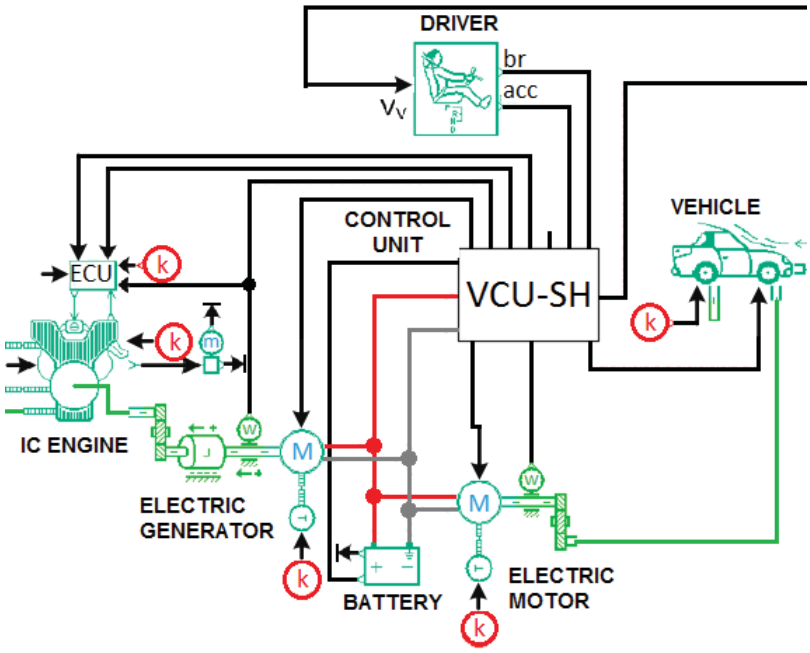


Figure 4.8: Synthesized `Series Hybrid` architecture simulation model.

The synthesized CPS simulation models enable domain engineers to validate their subsystems in software- and hardware-in-the-loop simulations, and to perform performance comparisons across various architectures. For example, Figure 4.9 uses the synthesized models for the `Series Hybrid` and the `Parallel Hybrid` to compare the State Of Charge (SOC) of the batteries during the first 600 seconds of the New European Driving Cycle (NEDC). Notice that the "`Convert RME to EE`" function in the `Series Hybrid` architecture in Figure 4.7 is mapped to the `Electric Generator` simulation component in Figure 4.8. This component allows the `Series Hybrid` architecture to increase the state of charge of the battery from 90% to 96%. In contrast, the `Parallel Hybrid` architecture does not specify the functionality for recharging the battery and therefore the SOC is reduced from 90% to 62%.

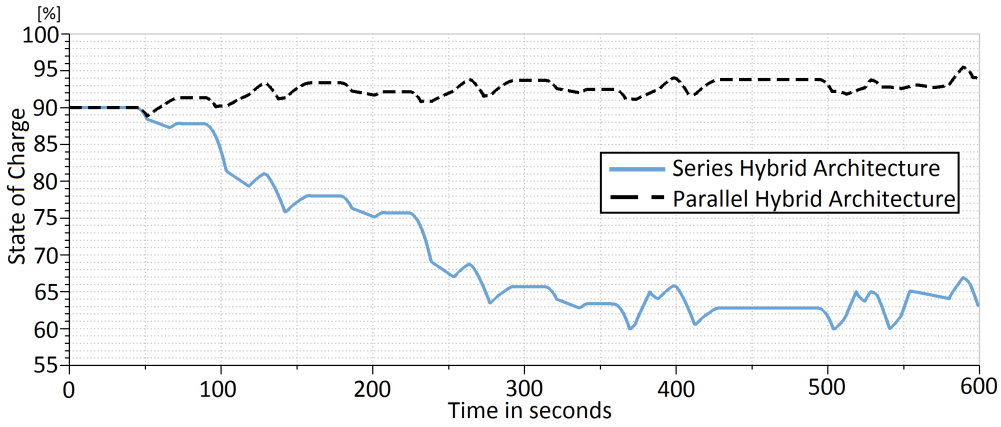


Figure 4.9: Battery's SOC among the `Series Hybrid`, and `Parallel Hybrid` architectures on the New European Driving Cycle (NEDC).

4.4.3 DSE Analysis using FMC

The automotive design example down-selects three architectures (`ICE`, `Series Hybrid`, and `Parallel Hybrid` architectures). In our experiments, we compare the synthesized simulation models and Figure 4.10 shows a comparison of the fuel consumption over 600s of the NEDC. The most interesting aspect to observe is that starting from a common functional model, the FMC enables the design space exploration of multiple automotive architectures through high-fidelity simulation models capable of answering both physics- and cyber-related engineering questions. It is also important to note that architectures are the enabling element for the FMC to synthesize accurate simulation models from functional models because an architecture is, after all, the allocation of functions to components. Architectures are the "recipes" for design and they play a central role in our novel CPS design methodology.

To illustrate the flexibility of the FMC consider the following scenario: a software engineer with marginal understanding of physical systems would like to develop a new control algorithm in an electric vehicle. A simple change in the functional model where the "`Fuel`" flow is replaced with an "`Electrical Energy`" flow guides the FMC to down-select the `Electric` and `Fuel Cell` architecture templates. The simulation model in Figure 4.11 shows the AMESim simulation model synthesized by the FMC for the `Electric` architecture template. Notice that Algorithm 5 selects `VCU-ELEC` component, an appropriate ECU

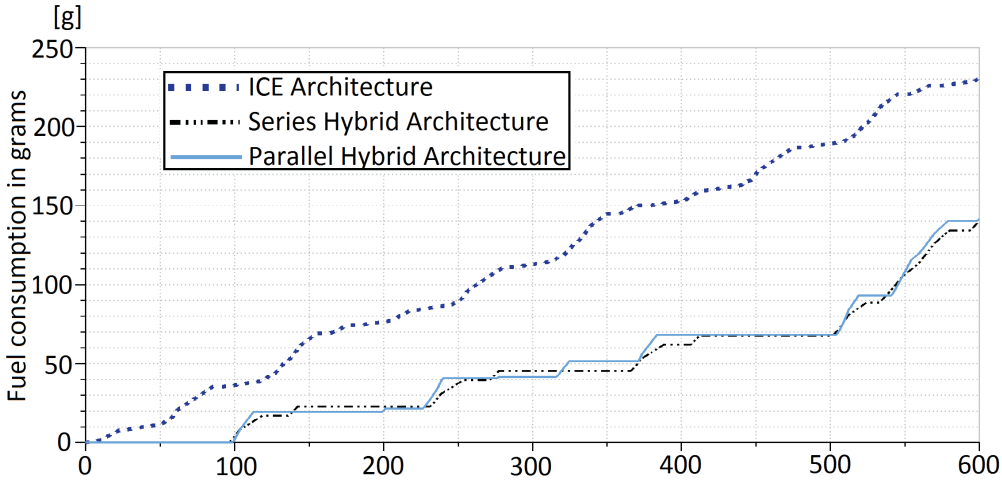


Figure 4.10: Fuel consumption comparison between the `ICE`, `Series Hybrid`, and `Parallel Hybrid` architectures.

for an electric vehicle. This simulation model provides the software engineer a realistic and high-fidelity multi-domain cyber-physical simulation model for software validation.

4.4.4 The Effectiveness of Feedback Function

Notice that in Figure 4.6, there are two "Feedback" functions which is proposed as a complementary contribution to the Functional Basis language in this paper. In order to demonstrate the effectiveness of the "Feedback" functions, we build another functional model of the engine system using only Functional Basis language (without "Feedback" functions), and then we synthesize this functional model using our FMC to generate the simulation models for the `ICE` architecture.

We compare the simulation results of both the simulation models synthesized from functional models with "Feedback" and without "Feedback" function. Figure 4.12 shows that with "Feedback" functions, the fuel consumption is much less compared to the simulation results without "Feedback" functions. Furthermore, much less emissions (CO, NOx, HC) are generated from the engine models synthesized with "Feedback" functions as shown in Figure 4.13. This is because, the "Feedback" function will be mapped to additional control

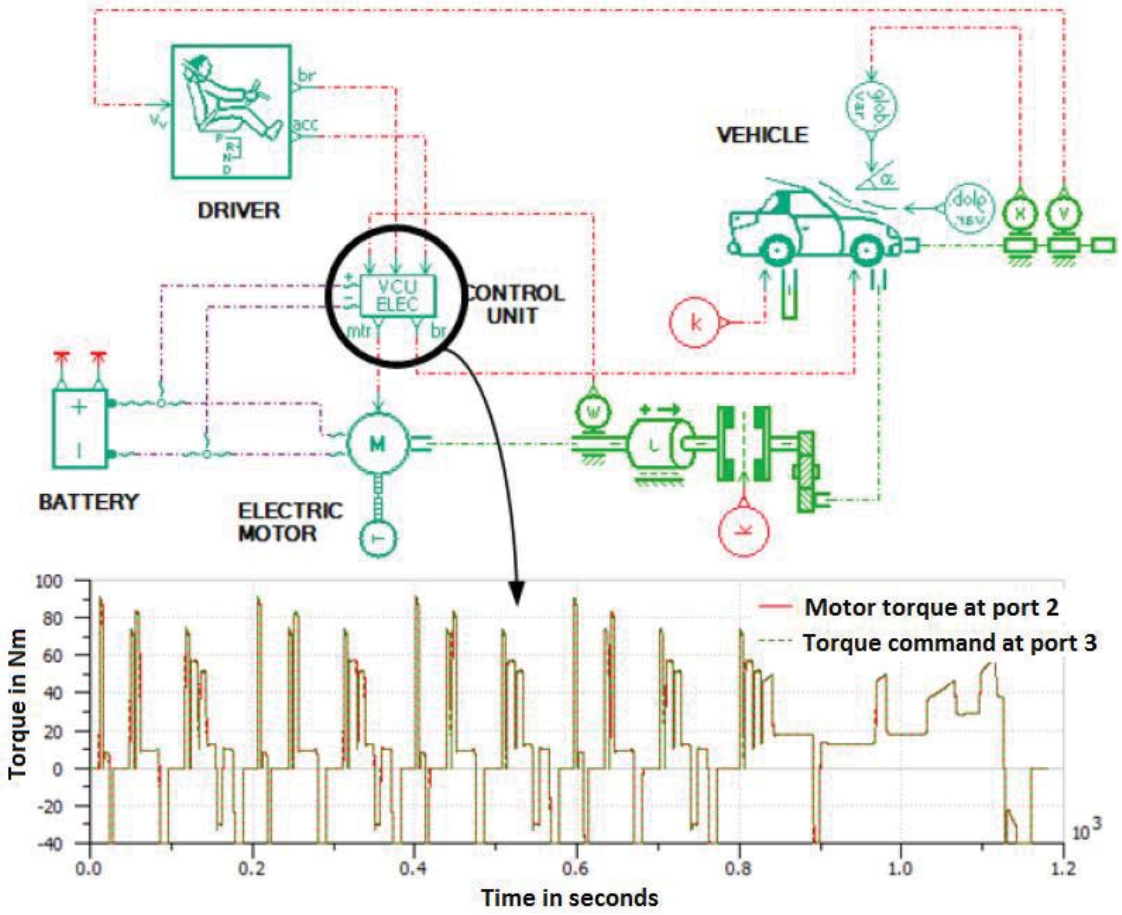


Figure 4.11: Synthesized simulation model for the `Electric` architecture.

units that could help to regulate the engine behavior much more efficiently. Clearly, the "Feedback" function can help the FMC to generate more practical multi-domain models.

4.4.5 Analysis of the Refinement Process

Figure 4.14 illustrates the bottom-up refinement capabilities of our methodology. The functions "`Convert RME to EE`", "`Convert EE to RME`", and "`Store EE`" do not exist in the original functional model in Figure 4.6. However, due to the partial mappings to architectures (`ICE`, `Hybrid`, `Electric`) generated from Algorithm 4, refined functional models are provided as suggestions to the system engineers. For example, given the mapping to the `Hybrid` architecture, new functions such as "`Convert RME to EE`", "`Store EE`" and

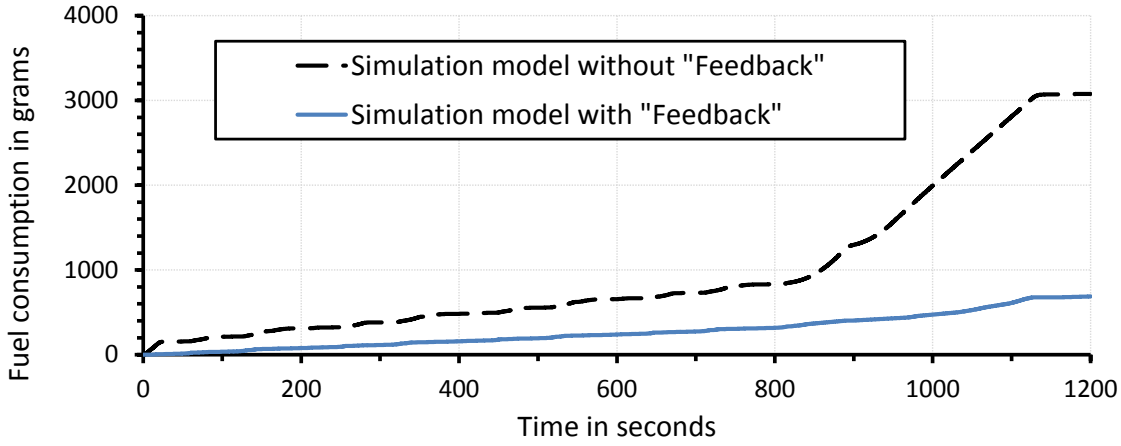


Figure 4.12: Comparison of the fuel consumptions between the synthesized simulation models with/without "Feedback".

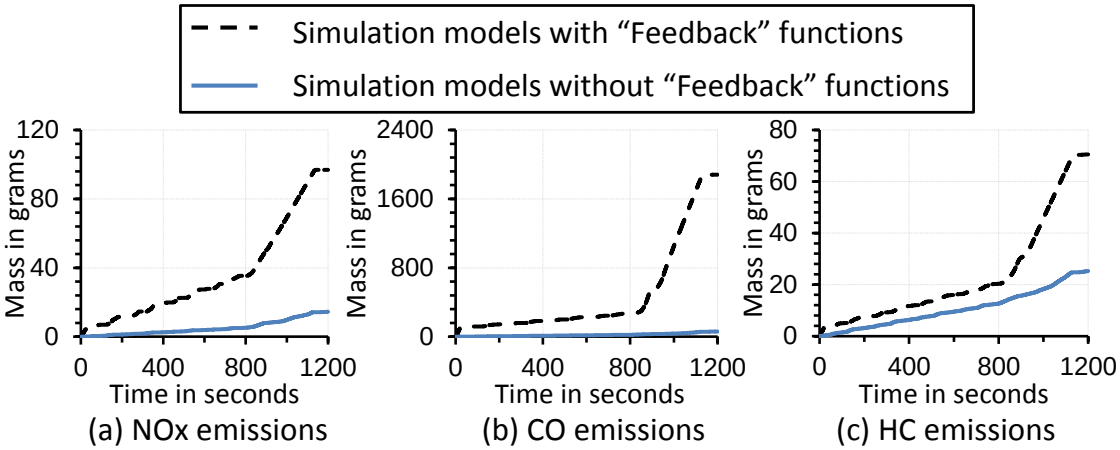


Figure 4.13: Comparison of emissions between the simulation models with/without "Feedback".

"`Convert EE to RME`" are created. Notice that thanks to Algorithm 5, this connection also exists in the simulation model in Figure 4.8. Figure 4.15 gives the number of functions in the refined functional models generated from three different mappings. Designers could use the refined functional models to iterate the whole design process for obtaining higher-fidelity models capable of answering more detailed questions. Therefore, our approach is complementary to platform-based design [151] and the *meet-in-the-middle* principle is satisfied.

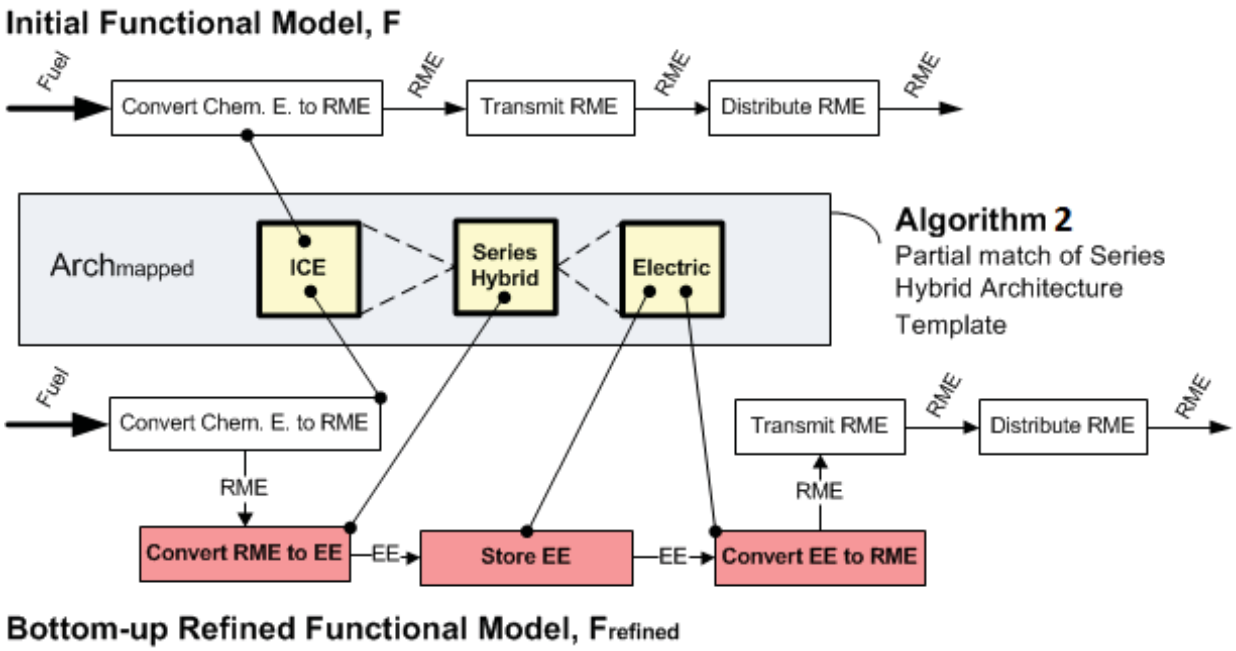


Figure 4.14: Bottom-up refinement of a functional model.

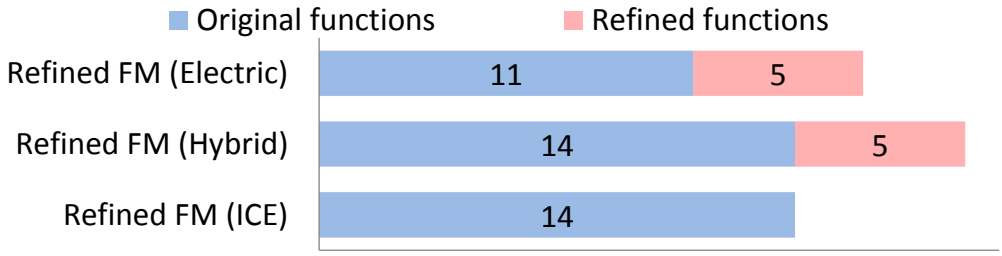


Figure 4.15: Number of functions in the refined functional models.

4.5 Conclusion

This paper presents a novel approach for the development of the automotive CPS starting from its functional model. Moreover, the paper presents a multi-disciplinary systems engineering methodology and associated design automation algorithms to automatically generate high fidelity multi-domain simulation models from the functional model. The presented FMC uses the already-known knowledge of architectures to develop alternative architecture-specific simulation models for such complex CPS and facilitate the early design space exploration. For the detailed implement of the FMC, we introduce the concept of a "Feedback" function to capture feedback flows that are essential for complex systems using control, software,

sensors, and actuators; a context sensitive mapping algorithm to construct an appropriate engineering context that facilitates the selection of function-to-component mappings by considering the surrounding flows of a function; and a refinement process to back-propagate the results as design suggestions to the system-level designers. Using an automotive engine system case study, we demonstrate the capability of FMC. Our experiments show that using the proposed FMC, various automotive architectures can be explored efficiently on major concerns such as fuel consumptions, emissions, battery status, and ECU-signals stability.

Chapter 5

CPS Co-Design at the Functional-Level

5.1 Introduction

Design decisions in one domain of CPS may completely change the constraints and requirements in the other domains, e.g. adding more functions in a modern automotive CPS may require changes to thousands of lines of software code or even the mechanical architecture. Existing CPS design methodologies are siloed in a specific domain and therefore have limited design space exploration capabilities, because only one domain can be tested at a time. This chapter presents a functional-level cyber-physical co-design methodology starting from the functional model of the CPS capable of concurrently expressing (multi-)physics and control in automotive applications. Moreover, we introduce a high-level synthesis algorithm capable of selecting a set of optimized system architectures using various executable simulation components and cost metrics. We demonstrate our methodology with a realistic automotive use case and explore various design alternatives for implementing the control systems in

pure continuous domain (e.g. traditional automotive subsystems without ECUs) or hybrid domain (e.g. Brake-By-Wire (BBW), Steer-By-Wire (SBW), Drive-By-Wire (DBW), etc.) under power, performance, and reliability constraints.

Automotive Cyber-Physical Systems (CPSs) are hybrid (continuous and discrete dynamics), heterogeneous (component variations), multi-domain systems consisting of various embedded systems, and physical processes connected through communication networks. The complexity in the modern automotive design is attributed, for the most part, to the software integration [38, 105, 137] during the engineering stage. Today, 90% of all the automotive innovations come from the electronics and software; and 40% of the development costs come from the Electronic/Electric (E/E) systems [62]. Unfortunately, the software integration in the automotive design is performed *after* the physical architecture has been decided. This is not an optimal method for various domains and also a counter-intuitive methodology for the design of CPSs because, in principle, CPSs are supposed to be tight integrations of "cyber" and "physical" processes. In our opinion, the state-of-the-art automotive design methodologies should be revisited in such a way that the software integration is validated as early as possible, and in parallel with the design of the rest of the system. This means that new design tools [48] should provide the cyber-physical design capabilities to fully exploit the benefits of software integration for increased performance, reliability, and decreased power, price, and time-to-market.

During the early design stage (aka concept design stage)[1], detecting and fixing errors is 5X to 10X less expensive and problematic than in the latter detailed design stages [61]. Therefore, it is highly desirable to validate CPS design as early as possible. However, the existing CPS design tools stay at the domain-specific levels [23, 53, 140] and this inhibits the possibility of appropriately handling the cross-cutting concerns that characterize CPS. For example, the early design of an automotive requires an experienced designer with a broad multi-

[1]In systems engineering, early design stage identifies the overall functionality and structure/architecture of a new system or product [35, 92].

domain knowledge to manually make decisions such as: should the control be implemented in a traditional continuous domain architecture or in an E/E hybrid-domain architecture? If E/E architecture is selected, what will be the minimum cost for the digital hardware implementation? What are the additional costs of sensors and actuators? Are additional continuous-domain electro-mechanical components required for implementing this control strategy?

State-of-the-art system-level automotive CPS co-design tools are currently missing, but are heavily desired by automotive industry [48]. Therefore, to answer these questions *as early and as accurately as possible in the early concept design stages*, this thesis proposes a new functional-level co-design methodology that emphasizes *breadth* – knowledge that spans across disciplines – rather than *depth* – knowledge that focuses on a particular domain – as it is the focus of the existing design tools. Our methodology starts with the functional models which formalize the requirements of a new automotive CPS functional models define, at the early design stage, what the system does in terms of energy, material, and signal flows. From a design automation perspective, this functional-level abstraction is a suitable formalism for CPS co-design because it decouples the intent (functionality) from its implementation (e.g., E/E vs electro-mechanical architectures). For example, an engine system's functionality is to convert chemical energy contained in the fuel into mechanical energy and hot gases. Given this functionality, several architectures, such as the traditional mechanical control system with driver and pedals or the modern E/E systems with many sensors and ECUs [105], from multiple domains may be applied. In this thesis, we demonstrate the applicability of our functional-level co-design methodology with a real automotive example to highlight the importance of having a system-level Design Space Exploration (DSE) capability during the early design stage of an automotive system. The novel contributions of this chapter are summarized as follows:

1. A functional-level co-design methodology for reducing the design time through the

DSE and identification of optimized architectures for the automotive CPS. This assists the designer to decide how, when, and what parts of the system should be integrated with software/cyber-based control to achieve the optimal system performance.

2. A modeling technique for partitioning control and (multi-)physics explicitly at the functional-level in order to support a fine-grain cyber-physical co-design for automotive applications.

3. A new high-level synthesis algorithm for supporting cyber-physical co-design from the cyber-physical functional models, where the control may be mapped to software-centric digital systems and/or conventional physical systems, to a set of optimized system architectures.

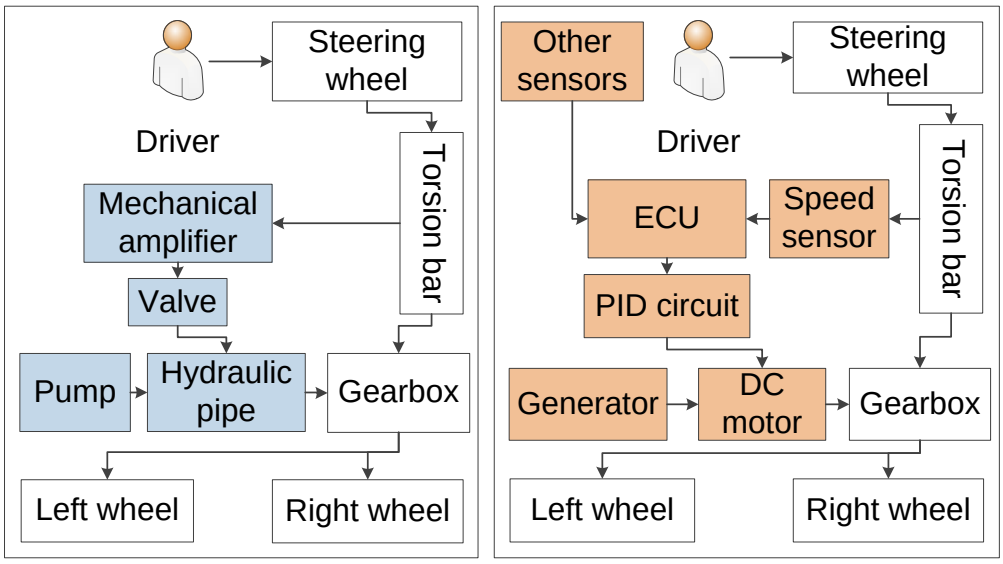
4. A companion set of cost metrics to verify the cost of the architecture models from different perspectives such as: performance, reliability, robustness, energy, weight, size, price, and emissions.

5. A complete case-study analysis using a real automotive application that demonstrates the advantages of our proposed co-design methodology.

The rest of the chapter is organized as follows: Section 5.2 uses a motivational automotive example to expose the need for early stage DSE capabilities. Section 5.3 introduces our functional-level co-design methodology. Section 8.7 presents an automotive DSE case-study. Section 5.5 discusses the main differences of our methodology relative to the related work. Finally, section 8.9 concludes the chapter.

5.2 Motivational Case-Study

One of the major challenges in designing a CPS is the integration of system-of-systems in such a way that performance and budget targets are met. For example, the steering system in an automotive is a critical sub-system with a relatively simple functionality; but it has dozens of different implementations. Figure 5.1 shows two typical architectures of the steering system: Figure 5.1 (a) shows the traditional Hydraulic Power-assisted Steering (HPS) architecture [156], and Figure 5.1 (b) shows the recent Steer-By-Wire (SBW) [89] architecture. In both architectures, a human driver is involved and acts as the main controller of the whole system. Furthermore, a *Torsion bar* connected to a *Gearbox*, and *two wheels* are common components in both architectures. The connections between the *Torsion bar*, *Gearbox*, and *two wheels* are mechanical connections since these components are all from the mechanical domain.



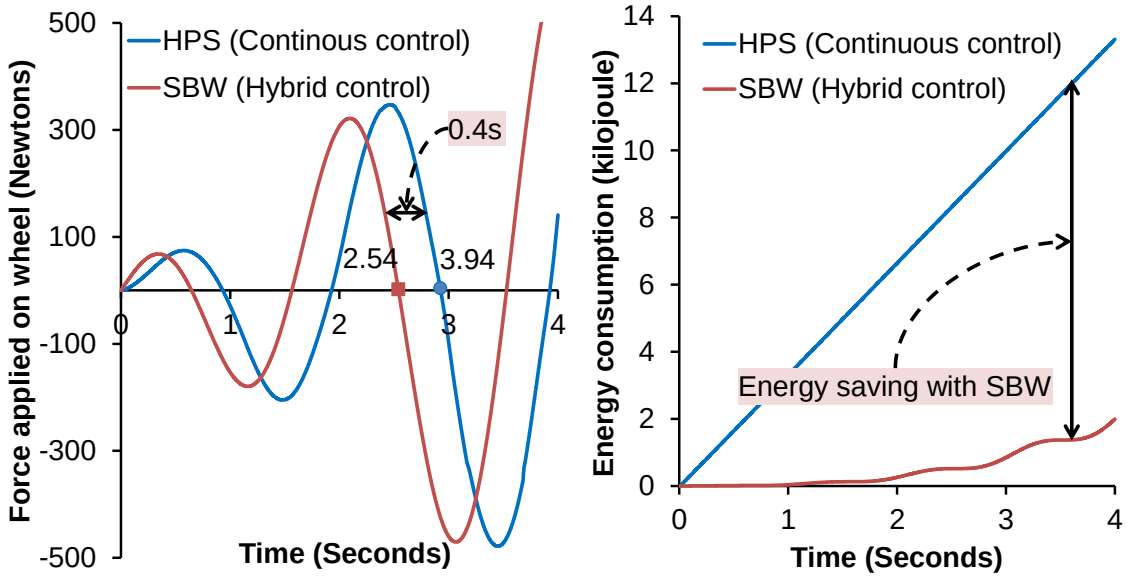
(a) Hydraulic Power assisted Steering (HPS) architecture

(b) E/E Steer-By-Wire system (SBW) architecture

Figure 5.1: Two example architectures for the CPS functional model of the automotive steering system.

The difference between these two architectures lies in how the assistance-power is provided.

In the HPS, a *Mechanical amplifier* and a *Pump* connected with a *Hydraulic pipe* provide the assistance-power. Since these are all mechanical domain components, most of the connections between these are mechanical connections without sensors and actuators. Only a *Valve* between the *Mechanical amplifier* and the *Hydraulic pipe* is required to act as the actuator that transfers a control signal in mechanical domain into hydraulic domain. In the SBW, an E/E subsystem with an *ECU* and a *Generator* connected to a *DC motor* provide the assistance-power. Moreover, a *Speed sensor* connects the *Torsion bar* and the *ECU* because control signals in the mechanical domain need to be transferred into E/E domain in order to control the *DC motor*. Finally, a *PID circuit* is implemented as an actuator to connect the *ECU* and the *DC motor* to convert control signals from the electronic domain to the electro-mechanical domain.



(a) Steering reaction-time (b) Total energy consumption

Figure 5.2: Performance and power consumption comparisons for the software-centric (SBW) and electro-mechanical-centric (HPS) automotive architectures.

These two architectures are compared to answer the design questions such as which architecture has better performance and less energy consumption. The performance is measured by the **reaction time**, which indicates how quickly a system responds to a change in its inputs. In the HPS, we observe that there is a major delay from the *Mechanical amplifier*

to the *Hydraulic pipe* because the mechanical devices have a relatively larger response time when compared to the electronic devices. In the SBW, the *Speed sensor*, *ECU*, and the *PID circuit* are electronic devices. Therefore, we observe that the SBW has a smaller reaction time. Experimental results[2] in Figure 5.2 (a) confirm that the HPS has $0.4s$ additional delay when compared to the SBW. Furthermore, we also notice that since in the HPS, the *Pump* is always working – even without a steering signal – it has a higher power consumption than the SBW. Experimental results in Figure 5.2 (b) shows that the HPS has high and roughly constant power consumption while the SBW has a more efficient power consumption.

Although the SBW has better performance, it may have significant reliability and safety concerns when compared to the HPS (a quantified analysis is provided in Table 5.3 in Section 8.7). Mechanical components, in practice, are considered error-free within the expected lifetime [38]. However the E/E components such as *ECUs* and *Speed sensors* may introduce potential failures that affect and propagate to the entire system. To minimize the safety concerns introduced by the E/E components, more reliable hardware and software are required, and these additional hardware and software consequently increase the overall system cost [137].

Today, it is well understood that the SBW systems have a higher performance but are more expensive and have a higher risk of safety-related issues (our experiments in Section 8.7 also validate similar results). However, as of today, there is no systematic method to decide between deploying software-based control and traditional non-software-based control in various sub-systems of an automotive system; nor are tools available to help designers quantify the costs (defined in Section 5.3.4) of such design alternatives through rapid DSE.

[2]Experiments are performed using the state-of-the-art automotive system simulation tool Amesim [1].

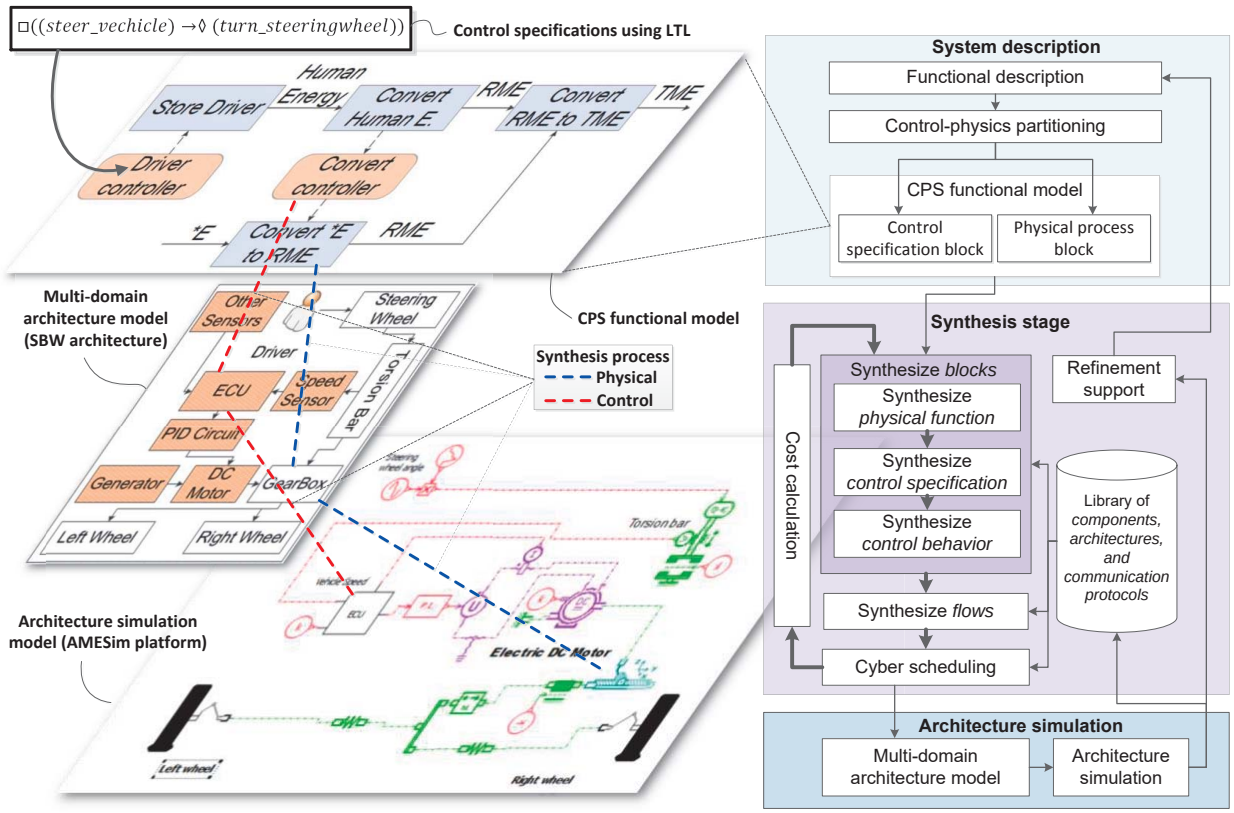


Figure 5.3: Proposed co-design methodology for automotive CPS. The right half of the figure describes the detailed steps of the proposed co-design methodology. The left half of the figure gives a synthesis example of a steering system (CPS functional model → Multi-domain architecture model → Simulation model).

5.3 Cyber-Physical Co-Design

From the example presented in Section 5.2, we analyze how the control behavior may be implemented in either the continuous domain (HPS system) or in the discrete domain (SBW system). There is a broad design space that allows for a variety of implementations of the control behaviors. In this section, we propose a computer-aided, functional-level co-design methodology that supports the DSE of the control behavior from a new level of abstraction (cyber-physical functional-level). Before going to the details of the design flow, we first provide several definitions that will help to explain our methodology.

5.3.1 Definitions

Definition 16. *A **functional model** is a formal description of the functional requirements for a system. It is defined as a labeled graph $FM = (F, E, L_F, L_E)$, where each node $f_i \in F$ is a function, each edge $e_{(i,j)} \in E$ represents the flow from f_i to f_j. Each function $f \in F$ and flow $e \in E$ have uniquely assigned labels $l(f) \in L_F$ and $l(e) \in L_E$ describing the type of the function f and flow e using a standard vocabulary.*

The vocabulary may be from any functional description language. In this work, for simplicity, the Functional Basis language [80] is used for describing the functions and flows.

A **CPS functional model** is an extension of the **functional model** that partitions and specifies the control and physical processes explicitly. The vocabulary used in CPS functional model may be the same as defined in the functional model. The detailed definitions of **CPS flow**, **control specification block** and **physical process block** are provided in Definitions 18,19,21.

Definition 17. *A **CPS functional model** is a formal description of both the control specification and the physical process of a system. It is defined as a labeled graph $CPS = \{C, P, E, A, L(C), L(P), L(E), L(A)\}$, where C is a set of control specification blocks, P is a set of physical process blocks, E is a set of CPS flows connecting C and P, and A is a set of architecture blocks. Each control specification block $c \in C$, physical process block $p \in P$, flow $e \in E$, and architecture $a \in A$ have uniquely assigned labels $l(c) \in L(C)$, $l(p) \in L(P)$, $l(e) \in L(E)$, and $l(a) \in L(A)$ describing their functions/specifications.*

Definition 18. *A **CPS flow** has three types: **signal**, **energy** and **material**. **Signal** flows are used to model the control-related signals such as the sensing-data and the control-commands. **Energy** and **material** flows are used to model the flows with real physical properties such as mechanical energy, electrical energy, fuel and gas, etc. For each flow, a label is attached to describe its detailed specification using a pre-defined vocabulary.*

Definition 19. *A **control specification block** models the control specification for a specific physical process. A **control specification block** takes signal flows (sensing-data and control-communication) as inputs and generates signal flows (control commands) as outputs. For each control specification block, a label is attached to describe its detailed specification using a pre-defined vocabulary.*

Definition 20. *A **control behavior** describes the detailed logic/behavior of a controller that satisfies a given control specification. Given a finite number of states X, inputs U and outputs V, it is defined as the function f that $f : (X, U) \rightarrow V$.*

The **control behavior** is used to describe the control algorithm in a **control specification block**. The function f can be described using various Models of Computation (MoC) such as finite-state machine or programming language.

Definition 21. *A **physical process block** models a real-world process on energy and materials. The description of the physical process follows the same vocabulary used in the functional model. The prime flow types of a physical process block are energy flow and material flow. Besides, a physical process block also receives control-commands inputs and provides sensing-data outputs in the form of signal flows. For each control specification block, a label is attached to describe its detailed functions using a pre-defined vocabulary.*

Definition 22. *An **architecture block** is a placeholder that is created during the early design stage. A label describing the target architecture of this block is given by the domain-specific engineers. It can only be implemented by the architecture having the same label.*

For example, an architecture block "*Steering wheel system*" is expected to be directly mapped to an existing "*Steering wheel system*" architecture. The vocabulary of the labels may be defined by the domain-specific experts manually.

Definition 23. *An **architecture model** is a synthesized system architecture. The architecture model is a graph $AM = (C, E)$, where C is architecture component mapped from the*

blocks in CPS functional model and E is the inter-components communication/connections mapped from the flows in CPS functional model. Moreover, each E has a label L(E) indicating the type of the communication/connections.

The type of the communication/connections can be from different domains, for example electronic communication protocols, mechanical connections, etc.

Definition 24. *A **cost** of an architecture arch is a value representing the penalty of an architecture on a specific perspective pes. The cost value* $Cost_{pes}^{arch}$ *is calculated based on a set of pre-defined cost models.*

In this chapter, we propose a set of cost models from several perspectives in Section 5.3.4. In the proposed cost model, bigger value of the cost indicates larger penalty on any perspective.

5.3.2 Co-Design Flow

The proposed co-design flow as presented in Figure 5.3 has three steps: 1) **system description**, 2) **synthesis stage**, and 3) **architecture simulation**. During the **system description**, a control-physics partition is specified to create a CPS functional model from the given functional models/specifications (see details in Section 5.3.3). The CPS functional model is the input for the **synthesis stage**, where blocks and flows are synthesized into models of components stored in the component libraries using the algorithm proposed in Section 5.3.5. The library consists of physical components including sensors/actuators, cyber components such as micro-controllers/communication networks, and existing architectures that may be directly mapped to *architecture blocks*. A *schedule* sub-step is involved to manage the scheduling of the control tasks and the communications[3]. When a set of *architecture models* are synthesized, the algorithm calculates and compares the costs of the

[3]Details of scheduling the control tasks are out of the scope of this chapter. However, we discuss the related work in Section 5.5.

architecture models from different perspectives (see details in Section 5.3.4). Based on the user-defined requirements, a set of architecture models are selected as the outputs of the synthesis stage to reduce the design space. Finally, in the **architecture simulation**, the generated architectures are simulated to validate that they fulfill all the given requirements. If the simulation results are satisfactory, the output architecture models are integrated into the library and refinement suggestions may be provided to the designers for further improving the CPS functional model.

5.3.3 Control-Physics Partition

Functional modeling has been shown to have the ability of modeling both the continuous and discrete domains at the same level of abstraction [32, 34, 80]. However these methodologies over-emphasize the modeling of physical processes and do not fully satisfy the modeling requirements for cyber-processes. To overcome these limitations and find a more appropriate formalism for the modeling of complex control systems that are integrated to multi-physics systems in the functional modeling, we propose to explicitly partition control and physics at the functional-level.

In our CPS functional model, we use two pre-defined types of blocks: 1) the *control specification block* and 2) the *physical process block*. During the system description step, both control and physical process blocks are defined. If the function involves a control behavior, a pair of blocks are defined: the control specification block c, and a physical process block p, with flows between them. If f does not require a control behavior, we simply convert f into a single p with the necessary flows. Additional flows required to connect all the blocks are also created during this step.

A control-physics partition example from the functional model to CPS functional model is presented in Figure 5.4. Notice that the syntax and semantics of the functions of a system

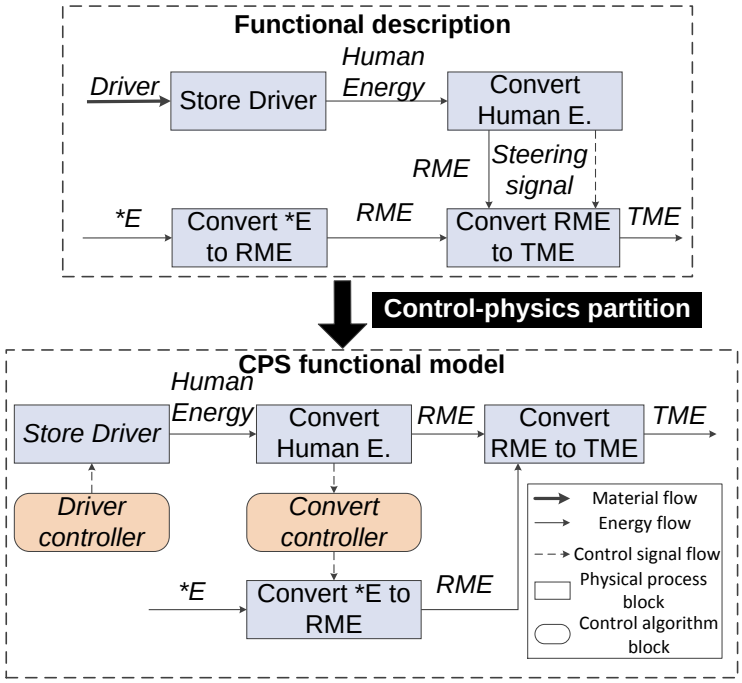


Figure 5.4: Control-physics partitioning of the steering system.

can be described using various existing functional modeling languages (Functional Basis language is used in this chapter). This example shows four functions: the function *Store driver* is partitioned into the *Store driver* and the *Driver controller*; the *Convert *E (Energy) to RME (Rotational Mechanical Energy)*[4] is partitioned into a physical process *Convert *E to RME* and the *Convert controller*. Notice that, in this example, other two functions, *Convert RME to TME (Translational Mechanical Energy)* and *Convert Human Energy*, do not need to be controlled, thus they are partitioned into physical processes without controllers.

Control Specification Description: To describe a *control behavior* (see Definition 20) requires an existing dynamic behavior (in the form of mathematical equations) of the physical process under control [20]. In order to model the control with only a high-level functional abstraction of the physical process, a method for describing the higher level control specification is required. Existing work [187] has shown that Linear Temporal Logic (LTL) may be used for the control specifications, and furthermore, tools and algorithms [147, 187] are avail-

[4]We use *E to denote any general types of energy, it means that *E may be in any form such as: mechanical, chemical, or biological energy.

able to synthesize LTL into control behavior described in the form of finite-state machine and/or other MoCs. In this chapter, we use LTL for the control specification.

Physical Process Description: As defined in Definition 21, to describe the *physical process block*, a high level language which may capture the domain-independent functions of a specific physical process is required. Existing work [32, 34, 179] have shown that the Functional Basis language is a suitable candidate for describing the physical process. As a result, in the scope of this work, we use the Functional Basis language to describe the physical process of the CPS functional model.

5.3.4 Cost Metrics

Selection of a specific architecture for an automotive design depends on various non-functional requirements and/or associated design costs. Therefore, the synthesis algorithm needs to automatically validate the system from different perspectives. Each perspective is mapped to the user requirements. For any valid design, the cost from any perspective must not violate the corresponding requirement. The validation for the cost $Cost_{pes}^{arch}$ of the architecture $arch$ depends on the cost functions $f_{pes}(C)$ on a specific perspective pes as shown in Equation 5.1, where C is the set of all the components in $arch$.

$$Cost_{pes}^{arch} = f_{pes}(C); \tag{5.1}$$

In our current work, we use various cost functions $f_{pes}(C)$. From the **performance** perspective, based on the execution time of the control tasks on the specific controllers and the reaction time of the physical process, f_{perf} calculates the total delay of each path in the graph of the architecture. We assume the number of paths in the graph is n and the delay of any component c_i is τ_i and $Cost_{perf}^{arch}$ is the largest number of all these delays as shown in

Equation 5.2.

$$Cost^{arch}_{perf} = max(\sum_{c_i \in path_0} \tau_i, \sum_{c_i \in path_n} \tau_i) \tag{5.2}$$

From the **reliability** perspective, assume that the failure rate of each component c_i in the architecture model $arch$ is a function of Mean Time Between Failure (MTBF) of each c_i: $FR_i(MTBF) = 1/MTBF$. We define $Cost^{arch}_{rel}$ as the failure rate of the whole system (see Equation 5.3).

$$Cost^{arch}_{rel} = 1 - \prod_{c_i \in arch} (1 - FR_i(MTBF)) \tag{5.3}$$

The **robustness** cost $Cost^{arch}_{robust}$ is used to indicate the quality of the whole architecture under different environments. We calculate the robustness cost based on the reliability cost, $Cost^{arch}_{rel}$. Given any environment E, the failure rate of each component c_i is modified to be environment depended as $FR_i(MTBF, E)$ and the reliability cost is a function of environment $Cost^{arch}_{rel}(E)$. Currently, $Cost^{arch}_{robust}$ is calculated by the ratio between two reliability costs under the harsh environment E_{harsh} and the normal environment E_{norm} (see Equations 5.4). E_{norm} and E_{harsh} are defined by the designers as requirements.

$$Cost^{arch}_{robust} = Cost^{arch}_{rel}(E_{harsh})/Cost^{arch}_{rel}(E_{norm}) \tag{5.4}$$

For the **power consumption** cost, $Cost^{arch}_{power}$, we identify all the energy flow inputs ϵ_i and outputs ϵ_o of the architecture. $Cost^{arch}_{power}$ is calculated using the power efficiency, defined as the total output power $Power_{\epsilon_o}$ divided by the total input power $Power_{\epsilon_i}$ as presented in Equation 5.5.

$$Cost^{arch}_{power} = Power_{\epsilon_o}/Power_{\epsilon_i} \tag{5.5}$$

For **price**, **weight**, **size**, and **emissions**, we model the costs for each component, c_i as a separate property p, w, s, e. Therefore, $Cost_{total}$ of the synthesized architecture is the sum of individual components on each perspective shown below:

$$Cost^{arch}_{price} = \sum_{c_i \in arch} p_i \tag{5.6}$$

$$Cost^{arch}_{weight} = \sum_{c_i \in arch} w_i \tag{5.7}$$

$$Cost^{arch}_{size} = \sum_{c_i \in arch} s_i \tag{5.8}$$

$$Cost^{arch}_{emission} = \sum_{c_i \in arch} e_i \tag{5.9}$$

5.3.5 Cyber-Physical Co-Synthesis Algorithm

In this subsection, we present our algorithm for synthesizing functional model into architectures in detail. The inputs to the cyber-physical co-synthesis algorithm include a CPS functional model CPS and a component library L including the existing models $Comp$ of the physical-components/sensors/actuators, pre-developed architectures $Arch$, and the cyber-centric communication protocols $Comm$. The output of the cyber-physical co-synthesis algorithm is a set of architecture models AM_{set}. The synthesis algorithm (see Algorithm 6)

consists of two steps. First, the *SynthesizeBlocks step* (see Algorithm 7) searches in the L to find all the possible $Comp$ which satisfy the functions described in the given CPS. Second, the *SynthesizeFlows step* (see Algorithm 8) maps all flows in the CPS to the sensor/actuator components in the L. After these two steps are completed, the algorithm evaluates the costs of all the synthesized $AM \in AM_{set}$ from different perspectives based on the cost functions introduced in Section 5.3.4. Any AM violating the given requirements R is excluded from the synthesized architecture models. The synthesis steps iterate until all the candidate mappings are found. If there exists no possible mapping (the returned $AM_{set} = \emptyset$), it means that the CPS functional model is not synthesizable and the designer may update the CPS based on the synthesized $Cost^{AM}$.

Algorithm 6: Cyber-physical co-synthesis algorithm.

Input: A CPS functional model: $CPS = \{C, P, E, A, LC, LP, LE, LA\}$
Input: Requirements: $R = \{req_1, req_2, ..., req_n\}$
Input: Component Library: L
Output: A set of architecture models: AM_{set}

```
1 SynthesizeBlocks(CPS, AM_set) SynthesizeFlows(CPS, AM_set)
2 foreach Cost perspective & (AM ∈ AM_set) do
3     Cost^AM_pes = f_pes(C)
4     if Cost^AM_pes can't satisfy R then
5         AM_set = AM_set/AM
```

Given the CPS functional model CPS as the input, the Algorithm 7 traverses through all the blocks in CPS to find appropriate $Comp \in L$ or $Arch \in L$ which satisfy the functions/specifications. Lines 3-10 find all the possible mappings from $A \in CPS$ to $Arch \in L$. Then, lines 11-18 find all $Comp \in L$ mapped from $P \in CPS$. Notice that line 14 validates whether the constraints of a component $Comp(CONS)$ (such as the input/output types, the functions achieved by the component, etc.) satisfy the physical functions described in $L(P) \in CPS$. Similarly, lines 19-29 find all $Comp \in L$ those satisfy the control specifications described in $L(C) \in CPS$. However, compared to the mapping of the physical processes, the control specifications [187] cannot be mapped directly to the constraints of components $Comp(CONS)$. Therefore, the mapping of $C \in CPS$ is divided into two sub-steps: 1) Line

Algorithm 7: SynthesizeBlocks(CPS, AM_{set}) step.

```
1  Variable: AM_temp, C_Behavior
2  AM_set = ∅
3  foreach A ∈ CPS do
4      AM_temp = ∅
5      foreach Arch ∈ L do
6          if (Arch ∈ L) matches (L(A) ∈ CPS) then
7              AM_temp = AM_temp ∪ Arch
8          else
9              Break
10     AM_set = AM_set × AM_temp
11 foreach P ∈ CPS do
12     AM_temp = ∅
13     foreach Comp ∈ L do
14         if Comp(CONS) satisfy (L(P) ∈ CPS) then
15             AM_temp = AM_temp ∪ Comp
16         else
17             Break
18     AM_set = AM_set × AM_temp
19 foreach C ∈ CPS do
20     AM_temp = ∅
21     Find P ∈ CPS controlled by C
22     foreach (Comp ∈ L) mapped to P do
23         C_Behavior=SynCtrlBehav(L(C), Comp)
24         foreach Comp ∈ L do
25             if Comp(CONS) satisfy C_Behavior then
26                 AM_temp = AM_temp ∪ Comp
27             else
28                 Break
29         AM_set = AM_set × AM_temp
30 return TRUE
```

23 calls the *SynCtrlBehav* function to synthesize $C \in CPS$ into a control behavior $C_{behavior}$ with the help of the behavior description of the physical components (e.g. mathematical models) and 2) Lines 24-28 check whether $C \in L$ satisfy the $C_{behavior}$ (such as the number of input/output signals, the complexity of the control algorithm, etc.). Finally, all the possible mappings are aggregated onto AM_{set}. Notice that the implementation of the function *SynCtrlBehav* is not in the scope of this chapter, existing work in [187] and [147] have provided the possible methodologies of automatically synthesizing control specifications into control behaviors described in LTL.

Algorithm 8 traverses $E \in CPS$ to satisfy all the design requirements. Lines 3-5 find all the

Algorithm 8: SynthesizeFlows(CPS, AM_{set}) step.

```
1  Variable: B_S, B_T, E_temp1, E_temp2, AM_temp
2  foreach E ∈ CPS do
3      B_S=Source block of E
4      B_T=Termination block of E
5      foreach ((Comp_s ∈ L) mapped to (B_S in AM_set)) && ((Comp_t ∈ L) mapped to (B_T in
       AM_set)) do
6          Arch_temp1 = ∅, Arch_temp2 = ∅
7          if (Comp_s & Comp_t) in the same domain then
8              foreach Comm ∈ L in the same domain do
9                  Set E_temp1 as a connection from Comp_s to Comp_t
10                 L(E_temp1) = Comm
11                 AM_temp=AM_temp ∪ EArch_temp1
12         else
13             foreach Comm ∈ L do
14                 if CONS(Comp) match the domains of Comp_s → Comp_t then
15                     foreach Comm ∈ L in the domain of Comp_s do
16                         Set E_temp1 as a connection from Comp_s to Comp
17                         L(E_temp1) = Comm
18                     foreach Comm ∈ L in the domain of Comp_t do
19                         Set E_temp2 as a connection from Comp to Comp_t
20                         L(E_temp2) = Comm
21                     AM_temp=AM_temp ∪ E_temp1 ∪ E_temp2 ∪ Comp
22         AM_set = AM_set × AM_temp
23 return TRUE
```

components for the source and destination block of $E \in CPS$. Lines 7-11 check whether the flow is connecting two components that are synthesized in the same domain. If the condition is satisfied, a single connection is constructed. Notice that if the domain is a cyber domain, Lines 8-10 will find all the possible communication protocols for this connection. Lines 13-21 handle the scenario when the source and the destination of the flow falls onto different domains. In such a scenario, a proper $C \in L$ is chosen to connect these two domains in Lines 13-14. Our exploration shows that this step may generate additional sensors and actuators for connecting components from different domains. Finally, Line 22 aggregates all the possible mappings.

In Algorithm 7, the loop at Lines 3-10 has a complexity of $O(n^2)$, the loop at Lines 11-18 has a complexity of $O(n^2)$, the loop at Lines 19-29 has a complexity of $O(n^3)$. Thus, the time complexity of Algorithm 7 is $O(n^3)$. In Algorithm 8, each of the inner loops at

Lines 8-11, Lines 15-17, and Lines 18-20 has a complexity of $O(n)$. Thus, the loop at Lines 13-21 has a complexity of $O(n^2)$ and the outer loop at Lines 2-22 has a complexity of $O(n^3)$. As a result, the time complexity of Algorithm 8 is $O(n^3)$. In Algorithm 6, the loop at Lines 3-6 has a complexity of $O(n^2)$. Therefore, the complexity of Algorithm 6 is also $O(n^3 + n^3 + n^2) = O(n^3)$, which is solvable in polynomial time.

5.4 Experiments and Case-Study Analysis

In order to demonstrate the capability of our co-design methodology in the automotive design, we present a case-study of designing part of an automotive system. The designed CPS functional model shown in Figure 5.5 is a partial view on several subsystems of an automotive. We assume that for this use case, four *architecture blocks* in Figure 5.5, *Driver*, *Engine system*, *Transmission system*, and *Wheel* already have the corresponding architectures in the library. Therefore, these blocks may be directly synthesized into existing architectures. However, the *Steering system* and *Braking system* do not have the corresponding architectures, therefore we have described these two subsystems in details using the CPS functional model. In this case-study, we use general energy type *E in designing the CPS functional model, therefore, this leaves the task of multi-domain DSE to the synthesis algorithm.

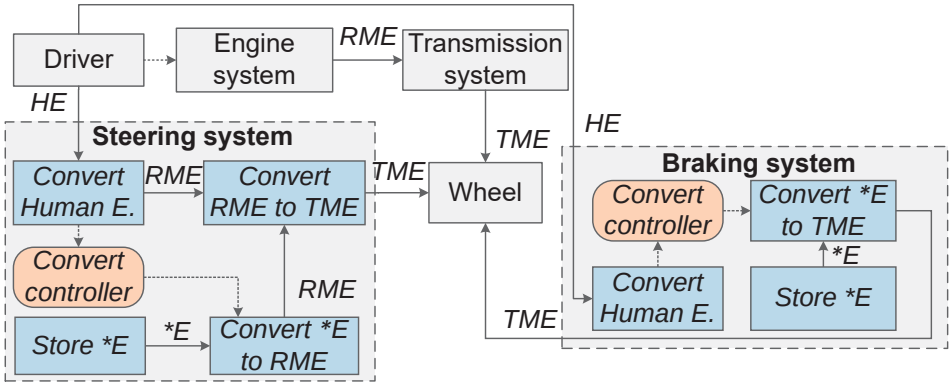


Figure 5.5: The CPS functional model for parts of an automotive.

5.4.1 Analysis of the Synthesis Process

After synthesizing the CPS functional model, we obtain five different architectures in this case-study. Notice that based on the available architectures from the automotive industry, we may get more synthesized architectures from this CPS functional model. However, in this chapter we only demonstrate five of those to verify our proposed methodology. The first architecture uses a purely traditional physical domain components; the second and third architectures use E/E components for braking and steering only; the fourth and fifth use the greatest number of E/E components for driving with varying types of different ECUs (single-core and multi-core). We denote these five architectures as: *Traditional/TRAD*, *Brake-By-Wire/BBW*, *Steer-By-Wire/SBW*, *Drive-By-Wire/DBW*, and *DBW (multi-core)*. Table 5.1 summarizes comparison of the number of required components among these architectures.

Table 5.1: Comparison of five synthesized architectures

	Comps numbers	ECUs numbers	Sensors numbers	Actuators numbers
TRAD	20	0	0	6
BBW	22	3	5	6
SBW	21	1	2	5
DBW	23	4	7	5
DBW (multi-core)	20	1	7	5

Our synthesis algorithm uses the parameters defined manually in the library (some components used for our experiments are shown in Table 5.2). These parameters are relevant for the cost function calculation and are implementation dependent[5]. Costs are calculated during the synthesis stage as presented in Section 5.3.5. Our synthesis algorithm shows that from **performance** and **power consumption** perspectives, *DBW* and *DBW (multi-core)* – with the greatest number of E/E components – represent the least expensive architectures, while the *TRAD* – without any E/E component – represents the most expensive architecture (3X more performance cost and 1.8X more power consumption costs compared to *DBW*

[5]Our parameters are representative of generic components.

and *DBW (multi-core)*). However, from **reliability** and **robustness** perspectives, *DBW* has 2.95X more the cost of the *TRAD*. Table 5.3 shows some of the cost calculation results using our synthesis algorithm. These results indicate that E/E components represent a trade-off between performance, power, and reliability. Notice that compared to *DBW*, *DBW (multi-core)* architecture maintain the same **performance** and **power consumption** while providing better reliability and robustness.

Table 5.2: Example list of the components in the component library

	$t_{reaction}$ $(seconds)$	P_{consum} $(watts)$	$MTBF_{norm}$ $(hours)$	$MTBF_{harsh}$ $(hours)$
ECU	<0.1	<100	1e+05	1e+04
DC Motor	<0.1	0-2500	1e+06	1e+06
Pump	<0.1	3000	1e+06	1e+06
Pipe	0.2	N/A	1e+06	1e+06
Gears	0.1	N/A	1e+06	1e+06

Table 5.3: Cost results for various automotive architectures

	$Cost_{perf}$ $(seconds)$	$Cost_{power}$	$Cost_{rel}$ $(errors/hour)$	$Cost_{robst}$
TRAD	0.3	51.4%	2e-05	1
BBW	<0.3	60%	4.9e-05	6.5
SBW	0.1	90%	3e-05	4
DBW	<0.1	90%	5.9e-05	7.1
DBW (multi-core)	<0.1	90%	3e-05	4

With the help of these cost calculations, the designer may obtain the optimized architecture according to the given requirements. For example, given a requirement that the performance cost should be less than $0.3s$ (for the purpose of better driving experience) and the reliability cost should be less than $4e-05$ $errors/hour$, *SBW* and *DBW (multi-core)* are the architectures those are satisfying these requirements.

In this case-study, we notice that the differences between two architectures, *DBW* and *DBW (multi-core)* lies in the cyber domain architecture. Figure 5.6 presents part of the architectures of the braking system for both *DBW* and *DBW (multi-core)* architectures. The convert controller (the control specification block in the braking system in Figure 5.5)

has the specifications described using LTL shown in Figure 5.6. The specification has three terms that may be synthesized onto three separate control tasks: Electronic Brakingforce Distribution (EBD), Electronic Stability Programme (ESP), and Anti-Lock Braking System (ABS). In *DBW* architecture, three single ECUs are deployed for each of the control tasks, whereas in *DBW (multi-core)*, all three tasks are implemented in a single multi-core ECU. The cost calculation results clearly show that *DBW* has 1.97X more cost on reliability and 1.78X more cost on robustness perspectives compared to *DBW (multi-core)*. However, both the two architectures have the same performance and power efficiency. Although in this case-study, we do not consider the other costs such as the price, weight those may be greatly effected by the cyber architecture due to the wiring problem and different sensor connections, our results still show that the integration of ECUs will bring greater benefit for the automotive design [143].

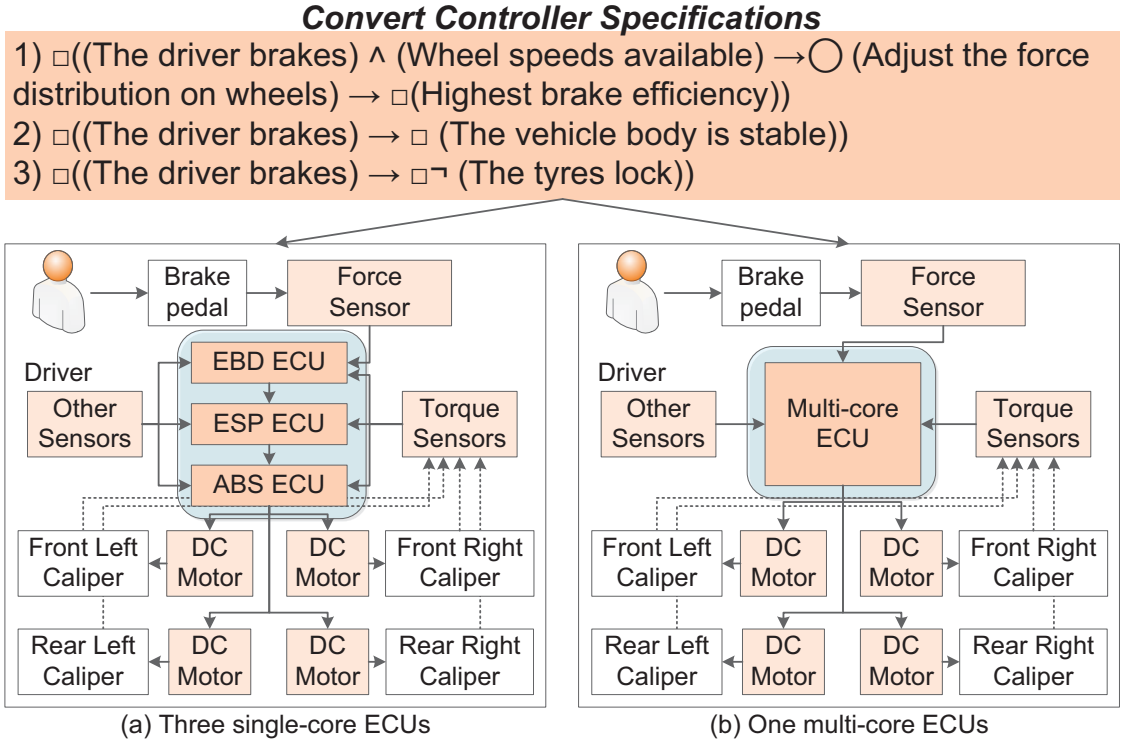


Figure 5.6: Comparison of the cyber domain architectures between DBW and DBW (multi-core).

5.4.2 Simulation and Validation

To validate the calculated costs of performance and power consumption, we consider a corner-turning scenario. The input signal is shown in Figure 5.7, where the driver first presses the braking pedal to slow down the automotive (the force of pressing the braking pedal is the input signal from time $0-1.5$ *Seconds*) and then turns the steering wheel (the angles of the steering wheel is the input signal from $1.5-3$ *Seconds*) to turn the automotive during a corner-turning scenario. We have implemented all these four architectures in the state-of-the-art multi-domain simulation platform Amesim [1]. The simulation results on performance and energy consumption shown in Figure 5.8 and 5.9 validate the cost calculation performed by our algorithm. For example, according to the cost calculation in the synthesis stage, the architectures *TRAD* and *BBW* use the hydraulic pipe for constructing the steering system and thus have a reaction delay of $0.2-0.3$ *Seconds*, which matches the simulation result in Figure 5.8.

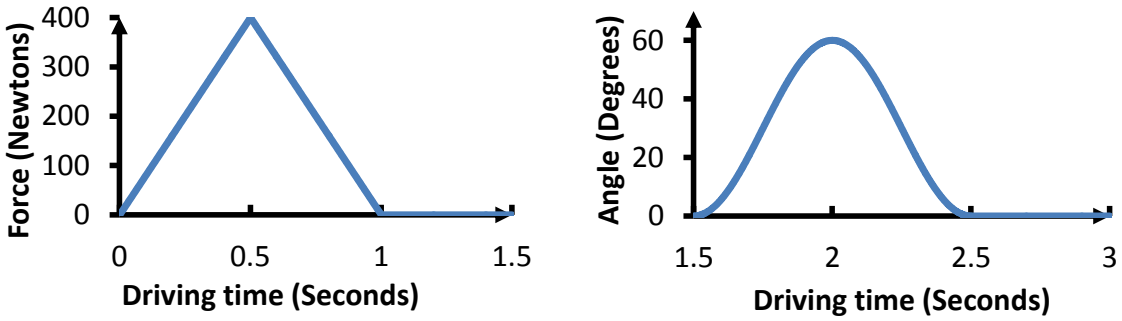


Figure 5.7: The input signals of a corner-turning scenario.

We have also conducted a tradeoff analysis between the reliability costs and the performance costs from the simulation results. The results presented in Figure 5.10 have validated our analysis in Table 5.3 during the synthesis process.

Although this case-study demonstrates the effectiveness of the proposed co-design methodology, some existing limitations need to be addressed in our future work. We currently have a limited number of components in the library. Therefore, adding more components could

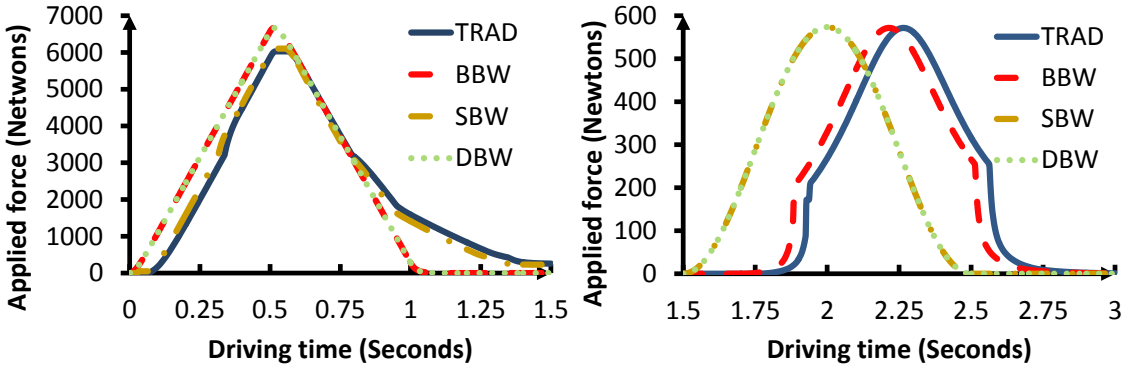


Figure 5.8: Performance comparison for various automotive architectures.

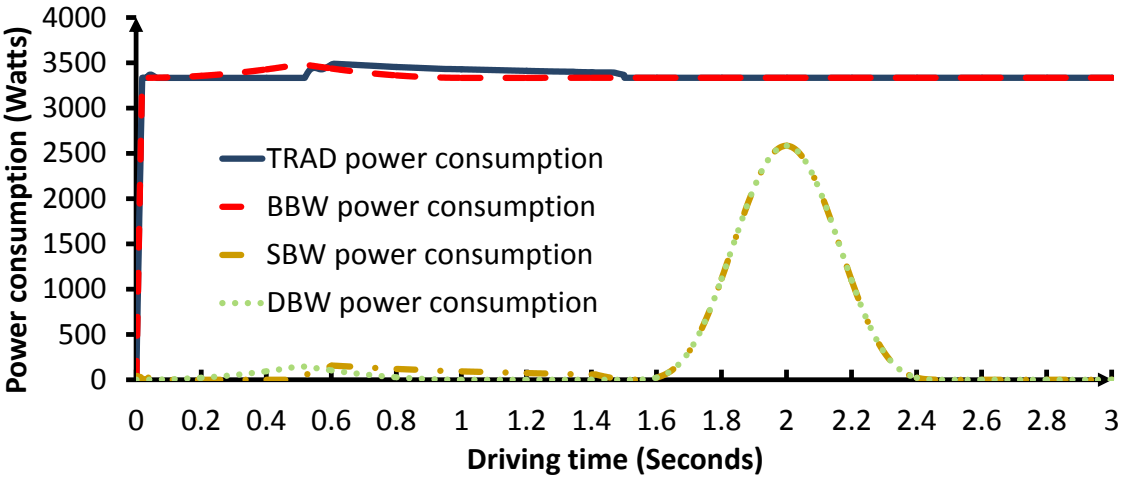


Figure 5.9: Power consumption comparison for various automotive architectures.

help the proposed methodology to handle more design tasks. Moreover, our CPS functional model description is currently limited to only Functional Basis language and LTL. Therefore, other functional modeling languages and tools for both cyber domain and physical domain description need to be evaluated in future.

5.5 Related Work

Efficient DSE in the semiconductor industry is a major area of research. Various Hardware/Software co-design methodologies and design automation tools [66, 72] have been proposed for identifying the optimized architecture for the given system objectives. Moreover, the co-design methodology has also been shown to be applicable for the multi-domain

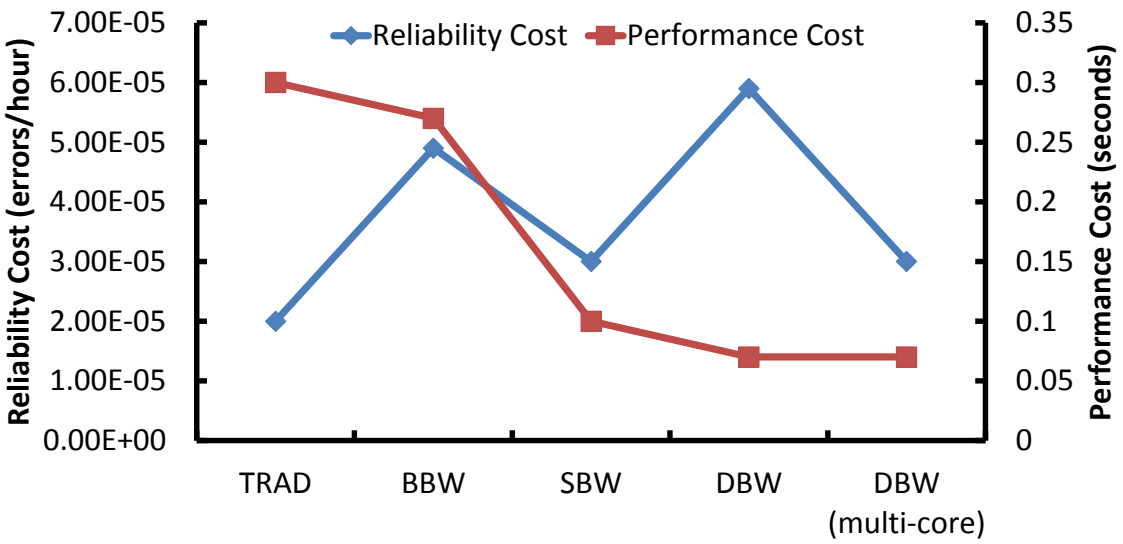


Figure 5.10: Tradeoff analysis between reliability and performance costs.

physical systems consisting of various physical domains such as electrical, mechanical, hydraulic, pneumatics, etc. [161]. In recent years, with the increasing interactions between the electronic-domain (control/computation/communication) and the physical domain, DSE across software-integrated multi-domains (continuous and discrete dynamics) has been considered as the grand research challenge for designing the complex CPS [164].

Although multi-domain design is addressed in the existing CPS design methods, most of them are limited in targeting the cyber parts or the physical parts separately. Compared to our work, there exists no method that is capable of providing a unified abstraction of the system that could explore the software-integrated multi-domain design space. Description language such as Architecture Analysis & Design Language (AADL) described in [60] is limited in describing the architectures for software. Modeling languages such as Modelica [171] or Simscape [112] are mainly proposed for modeling the multi-domain physical components. The LTL specifications proposed in [187] is specialized for control specification. For example, Pessoa [114] is a tool capable of synthesizing the embedded controllers from the control specifications. Work in [32, 34, 35, 179, 185] including our prior work provide methods for synthesize functional models into simulation models; however they are mainly focused on physical components. Our prior work in [35, 179] explore the existing architectures

for synthesizing functional models into simulation models; however, they are limited on physical architectures only. Moreover, functional modeling technique shown in our prior work in [35, 179] is not capable of partitioning control and (multi-)physics explicitly at the functional-level in order to support a fine-grained cyber-physical co-design. Work in [8] have demonstrated the importance of multi-domain design for a CPS in the system-level. However, [8] does not provide a solution for automatically generating architecture models or detailed designs from the functional models.

Furthermore, targeting multi-domain CPS designs, Model-Based Designs (MBD) [53], Concurrent Engineering (CE) [169] are proposed for the purpose of reducing design time and exploring the design space. However, all these works mainly rely on the knowledge of the domain experts to make the decisions during the modeling stages. Therefore, this limits the DSE compared to our domain-independent functional modeling methodology. Another methodology in [134] named Multi-View Model (MVM) is proposed for designing multi-view systems. Compared to our high-level synthesis methodology, this work focuses on describing the relationship between multi-views (e.g. control view, software view, architecture view etc.) of a design.

Besides the DSE for architectures, scheduling of communication and computation is another major research area in the automotive CPS co-design. Works presented in [67, 69, 148, 193] provide various methodologies for control-communication co-design of the CPS. In this chapter, we focused on presenting the synthesis stage and DSE of various automotive architectures. Although we focus on presenting the synthesis stage and DSE of various automotive architectures, these existing scheduling works may help our method to generate more realistic simulation models after the architecture is decided.

5.6 Conclusion

This chapter proposes a novel functional-level co-design methodology for the automotive CPS. The marriage of systems engineering principles with high-level synthesis techniques from design automation area results in a new co-design methodology capable of helping the DSE and validation of multi-domain automotive architectures. A new modeling technique for the CPS, referred to as the CPS functional model, is presented to capture the high level functions and specifications of (multi-)physics and controls. Starting from the functional model, a synthesis algorithm is presented to generate the detailed architectures and associated costs on different perspectives for the automotive CPS. Furthermore, the architectures can be translated to simulation models using various executable components from multi-domain simulation platforms. Using the proposed methodology, an automotive case-study has been demonstrated to explore five alternative architectures. Our synthesis algorithm shows that: *TRAD* has 3X more the performance cost and 1.8X more the power consumption costs compared to *DBW*, while from the reliability and robustness perspectives, *DBW* has 2.95X more costs compared to *TRAD*. Moreover, our co-design methodology and the cost analysis have been validated using the state-of-the-art multi-domain automotive simulation platform.

Chapter 6

System Level CPS design using Non-Euclidean Training method

6.1 Introduction

One of the key challenges in design automation of CPS is that some of the system level critical design decisions still require engineers' manual efforts. The reason is that most of these critical decisions requires creative thinking rather than logical behaviors which can be automatically explored by machines. Specifically, when a functionality of part of a CPS is given, engineers usually need to manually make system-level decisions such as choose the architectures of the system or choose the physical domains to implement the functionality. Fortunately, thanks to the recent development of machine learning technology, the engineering knowledge can be learned by machines to enable design automation of CPS from its functional perspective.

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific

task) with data, without being explicitly programmed [150]. Artificial neural network (ANN) is one of the most commonly used models to implement the machine learning techniques. With the growth of computational power in the past few years, training an ANN model with many layers is possible. This method of training a multi-layered ANN is also known as deep learning. Deep learning has achieved a great success in solving problems such as speech recognition, image processing and computer vision. This success, however, are for clustering and classification of grid-structured Euclidean data (e.g. 1-D time-series signals, 2-D images, etc.). For example, Convolutional Neural Network (CNN) is a special ANN which specifically works on 2-D images, which can be considered as functions on the Euclidean space, sampled on a grid [22].

However, many of the real-world problems require learning tasks based on data which does not have a underlying Euclidean structure, defined as non-Euclidean data. Graphs and geometric manifolds are the two most common types of non-Euclidean data. For example, in social networks, the data is modeled as a social graph in which the users' information are embedded on the vertices, and the relations between users are embedded on the edges [46, 88, 174]. In Cyber Physical Systems (CPS) or Internet of Things (IoT), system components such as sensors, actuators and controllers can also be modeled as a system graph, and the status of these components can be modeled as time-dependent signals on the vertices. In information system and software system analysis, a data flow diagram (DFD) is also a non-Euclidean graphical representation of the "flow" of data through the system [26]. In electronic design, a netlist is a special type of graph describing the connectivity of an electronic circuit, which consists of a list of the electronic components in a circuit and a list of the nodes they are connected to. In computer graphics and vision, 3D objects are modeled as Riemannian manifolds (surfaces) which defines various geometric notions such as angles, lengths of curves, areas (or volumes), curvature, gradients of functions and divergence of vector fields. [87]. In molecular biology, Protein Graph Repository (PGR) aims to explore the protein 3D-structure within a graph perspective where a protein is seen as a network of

amino acids [54].

To replicate the same success of machine learning on Euclidean data, training the non-Euclidean data faces many challenges such as: lack of common system of coordinates, no vector space structure, shift-variance, and rotation-variance. As a result, the basic operations in ANN like convolution and pooling need to be redefined on non-Euclidean domains. And eventually, a new type of ANN needs to be developed for non-Euclidean data structure. However, to solve the different types of non-Euclidean data (e.g. Graph and Geometric manifolds) training problems may require different types of ANNs. In this chapter, we are trying to a make step forward toward design automation of CPS with machine learning technique, and we are only targeting the problem of learning non-Euclidean data in the engineering domain. We assume the underlying data structure is a graph. Thus, we propose a Structured Graph Convolutional Neural Network (SGCNN) model that can be used to learn graph structured data. Nevertheless, the proposed model may also be applied in domains which require learning graph-structured data.

6.2 Related Works

Recently, researchers have been trying to apply CNN for non-Euclidean structured data. Briefly, we classify two types of non-Euclidean data learning problems in the existing works: (1) Learning and extracting features about the structure of the data; (2) Learning and analyzing the functions of the signals of the data. Most of the existing works for geometric manifold learning are trying to solve the first problem [16, 36, 52, 74, 79, 90, 145, 168]. They try to project a local high dimension structure into a low dimension space (e.g. a local coordinate system, an eigenvector based vector space) and extract features from the low dimension space for learning purposes.

In this chapter, for the engineering domain problem, our interest falls into the second type of the problems. **Specifically, we are trying to classify existing CPS design data, which is represented in the form of graphs, into useful functions.** By learning the functions from existing CPS designs, the machine can extract knowledge from human design experience which is usable for high level design automation.

There have been multiple attempts to apply CNN on graphs. In general, there are two different approaches to achieve it. One is based on spectral domain analysis, the other is based on spatial/vertex domain analysis. In [52, 90], a Fourier transformation on graphs is proposed to project the high dimension signals, which lives on the vertex of the graph, to low dimension space constructed by the eigenbasis of the graph Laplacian operator [44]. Approximation of the transformation is also proposed in the current work to reduce the complexity. However, the spectral domain approach has a major limitation: it is not graph invariant. This is because all the spectral domain approaches rely on the Laplacian matrix of the graph. In other words, the Fourier transform on different graph will be different (due to the eigenvectors of Laplacian matrix being graph-dependent), thus a CNN trained for one graph can't be applied on another one.

On the other hand, most of the vertex domain approaches are based on the aggregation of neighbor nodes' information for every node in the graph [71, 75, 157], thus they are graph invariant. For example, in the recent GraphSAGE work [75], the authors propose to train a function to sample and aggregate a node's local neighborhood to the center node. In this work, the samples are generated in a breadth-first search manner. Similarly, there also exists other works which differ mostly at the sampling methods. For example, in [71], the authors proposed to use a factor to tune the sampling ratio between breadth-first search and depth-first search. The vertex domain approach has shown the effectiveness on node-level clustering and classification. However, it also has the limitation, which is that it only works on node-level. And in many real-world problem, we want to be able to cluster or classify a

whole graph or subgraph instead of a single node (e.g. 3D CAD model, social networks). Making an analogy to image classification, we want to classify whole images rather than pixels.

In addition to CNN on graph, traditionally graph kernels have been used in structure mining to measure the similarity of pairs of graphs [177]. Graph kernel can be used to classify or cluster graphs and subgraphs, however, it only considers the structure similarity between pairs of graphs. In engineering domain, it is not necessary for two subgraphs with different structures to be in different clusters. For example, two 3D-CAD models of an automotive system may have different structures but they will still have the same function and be in the same cluster labeled as automotive.

6.3 Contributions

In this chapter, we propose a Structured Graph Convolutional Neural Network (SGCNN) that is able to perform graph invariant learning tasks at graph and subgraph level. The proposed SGCNN will be used to help the CPS design automation in the engineering domain. The major contribution of SGCNN are as follows:

- A novel SGCNN architecture that is able to perform training tasks at graph/subgraph level.
- A new path-based neighbor nodes aggregation method that aggregates vertex domain neighbor nodes information onto subgraphs.
- A novel subgraph convolution kernel that can perform graph invariant convolution operations on graph or subgraph.
- A downsampling/pooling algorithm based on the local structure of the subgraph.

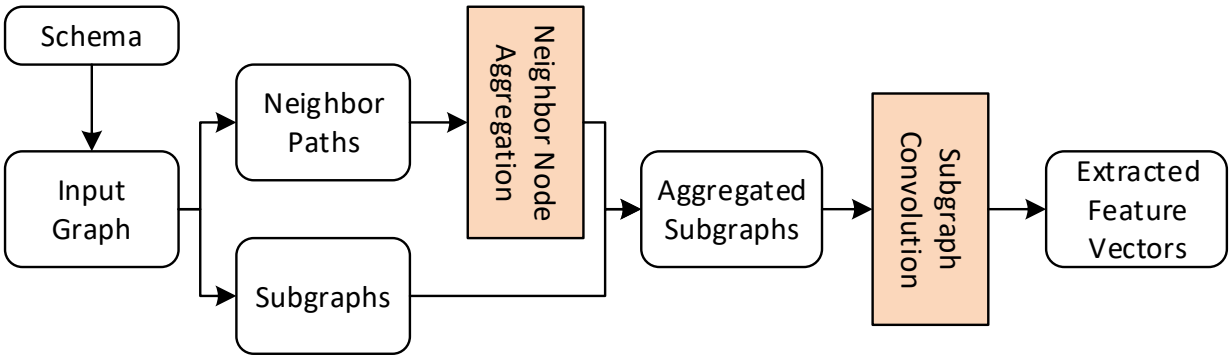


Figure 6.1: SGCNN Architecture

6.4 SGCNN Architecture

We define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. The graph edges can be weighted and directed. However, for simplicity, we only consider unweighted graph in this chapter. For each $v_i \in \mathcal{V}$, we define the features to be f_i. And we define the adjacency matrix of $\mathcal{G}$ to be $\bar{A}$. A subgraph is defined as $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, where $\mathcal{V}_s \in \mathcal{V}$ and $\mathcal{E}_s \in \mathcal{E}$.

The proposed SGCNN Architecture is shown in Figure 6.1. We assume a *schema* is available to generate subgraphs from the input graph. The *schema* should be given by the user as an input, which can be used for graph query [188]. It has two major steps (highlighted in Figure 6.1): (1) Neighbor Nodes Aggregation implements a path-based ANN to aggregate neighbor nodes information into the target subgraph. (2) Subgraph Convolution Kernel implements a graph invariant CNN on the target subgraph to extract feature vectors. The details will be presented in the following sections.

6.4.1 Neighbor Nodes Aggregation

Similar to the vertex domain approach in [75], given a graph $\mathcal{G}$, we propose to aggregate the neighbor nodes features of the target subgraph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. By considering that there

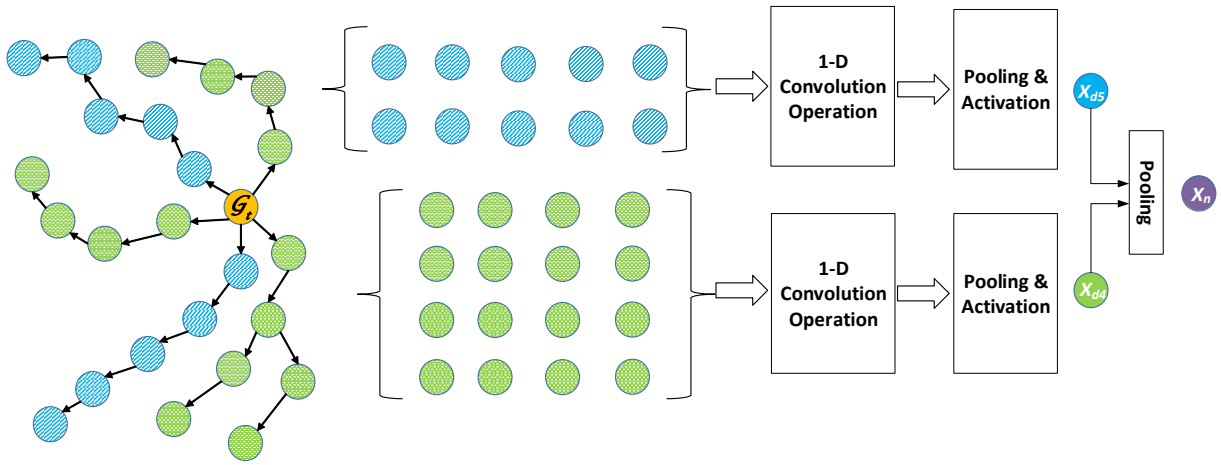


Figure 6.2: Neighbor Nodes Aggregation Example: Yellow node is a simple representation of a subgraph $\mathcal{G}_t$; Blue nodes represent length 5 paths; Green nodes represents length 4 paths.

are two methods to collect neighbor nodes in a graph: (1) Breadth first search; (2) Depth first search, we proposed a generalized path-based approach that integrates both of these two methods in engineering domain data. Specifically, we have developed the Neighbor Nodes Aggregation Layer which serves as hidden layers in SGCNN. For an Neighbor Nodes Aggregation Layer l_{ag}, we define two parameters: d as the depth to search, and n as the number of paths to be computed.

For all $v_i \in \mathcal{V}_t$, we search $\mathcal{G}$ to find all length d paths $\mathcal{P}_i^d$, which includes v_i, but doesn't include any other nodes in $\mathcal{V}_t$. Then we get all the length d neighbor paths of $\mathcal{G}_t$ as $\mathcal{P}^d = \{\mathcal{P}_0^d \mathcal{P}_1^d, ...\}$. From $\mathcal{P}^d$, we randomly select s paths (e.g. we can uniformly draw s samples), where s is a input parameter, and use each path as a row to form a neighbor feature matrix $\bar{N}$. Thus $\bar{N}$ is a n by d matrix with each element being a feature vector of a neighbor node. An example of this step is shown in Figure 6.2. Notice that in case the number of paths found in $\mathcal{P}^d$ is smaller than n, we can simply pad the $\mathcal{P}^d$ to make $\bar{N}$ at least have n number of rows/paths.

Next task is to extract feature vectors from $\bar{N}$ as the output of this layer. As discussed in [75], the non-Euclidean data such as graph has no nature ordering. Thus, the feature extraction needs to be applied over an unordered set of paths. Or in other words, the rows of matrix $\bar{N}$ can be arbitrarily exchanged, and the extracted features should remain unchanged. In

this chapter, We apply a general 1-D convolution operation with a trainable 1 by d weight matrix $\bar{W}$ on $\bar{N}$ firstly, then we use a symmetric pooling function to extract the neighbor nodes feature vectors x_n as follows:

$$x_n = \sigma(f_{pool}(\bar{W} \circledast \bar{N}) + b) \tag{6.1}$$

Where, b is a bias variable, σ is an activation function (e.g. ReLU function), and f_{pool} is a pooling function which is invariant to permutations of rows in $\bar{N}$. For example, f_{pool} can be a mean operator over all elements in the matrix, or over all the rows in the matrix. Similarly, f_{pool} can also be a max operator as well. We argue that different f_{pool} functions/operators may be suitable in different domains, it can be a configurable parameter during the training process.

Notice that, we can aggregate paths with different lengths. For example, as shown in Figure 6.2, features from both length 5 and length 4 paths are extracted. In general, assume we extract features from paths with lengths $\{d_1, d_2, d_3, ..., d_k\}$ as $\{x_{d1}, x_{d2}, x_{d3}, ..., x_{dk}\}$. A pooling function f^d_{pool} can also be applied to reduce the dimensions of extracted features as:

$$x_n = f^d_{pool}(\{x_{d1}, x_{d2}, x_{d3}, ..., x_{dk}\}) \tag{6.2}$$

Finally, we will aggregate x_n to $\mathcal{G}_t$ by concatenate all the feature vectors of $v_i \in \mathcal{V}_t$ as $x_{agg} = \{x_i, x_n\}$. Algorithm 9 summaries our Neighbor Nodes Aggregation process.

Algorithm 9: Neighbor Node Aggregation Process.

Input: An input Graph: G
Input: A schema for query: $Schema$
Input: A list of depth to search: $D = d_1, d_2, ..., d_n$
Input: A list of sample numbers per depth: $S = s_1, s_2, ..., s_n$
Output: A feature vector: x_n

```
Query G with Schema to generate subgraphs G_t
foreach d_i ∈ D do
    foreach v_j ∈ V_t do
        Find all length d_i paths P_j^{d_i}
        Remove paths contains nodes in V_t from P_j^{d_i}
        Add P_j^{d_i} into P^{d_i}
    Construct N̄ by randomly select s_i paths from P^{d_i}
    Extract feature x_di from N̄
x_n = f_pool^d(x_d1, x_d1, ..., x_dn)
return x_n
```

6.4.2 Subgraph Convolution Kernel

The task of our proposed Subgraph Convolution Kernel is to extract feature vectors from graph or subgraph. First, we define the attribute matrix of a graph or subgraph as follows: Given a target graph $\mathcal{G}$ with n number of vertices, by listing all the feature vectors x_i , which lives in $v_i \in \mathcal{V}$, in any given order, we can get the list of feature vector $\mathcal{X}$. Then, we repeat $\mathcal{X}$ by n times to form the feature matrix $\bar{X}$ with each row being $\mathcal{X}$. With the adjacency matrix $\bar{A}$, we define the attribute matrix $\bar{A}r$ of $\mathcal{G}$ as the Hadamard Product between $\bar{X}$ and $\bar{A} + \bar{I}$ as follows:

$$\bar{A}r = \bar{X} \circ (\bar{A} + \bar{I}) \tag{6.3}$$

where $\bar{I}$ is the identity matrix. For the purpose of maintaining the feature vectors of every vertex without loosing their own information [90], we add a self-loop to each node by the addition of $\bar{I}$. Notice that although we discuss only about unweighted graph in this chapter, the proposed work can be expanded to weighted graph by applying a weighted adjacency

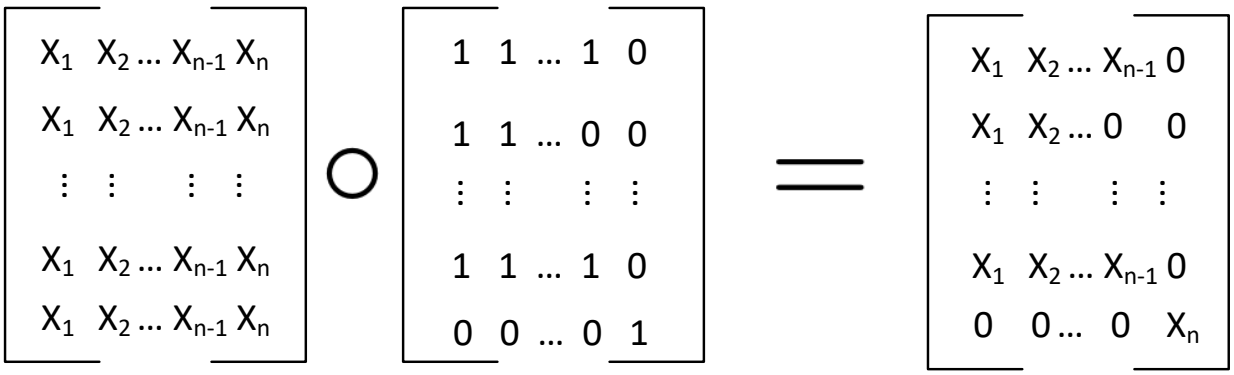


Figure 6.3: Attribute Matrix Example

matrix $\bar{A}$. An example of attribute matrix is shown in Figure 6.3.

Taking $\bar{A}r$ as the inputs for convolution operation on graph, we define a graph convolution kernel to be a k by k weight matrix $\bar{W}^k$. Then, we will apply convolution between $\bar{W}^k$ and $\bar{A}r$. However, different from a grid-structured 2D convolution between matrices (which the kernel will slide following a top-down, left to right order), we propose a new definition of convolution operation in graph data structure.

Since each row or column in $\bar{A}r$ is actually corresponding to a vertex in $\mathcal{G}_t$, thus, removing the i row and i column equals to remove the vertex i from $\mathcal{G}_t$. Assume n is bigger than k, we propose to remove $n-k$ number of vertex from $\mathcal{G}_t$, and the left over subgraph has a new k by k attribute matrix $\bar{A}r^k$. And there is $\binom{n}{k}$ number of possible $\bar{A}r^k$. Thus, for all the possible $\bar{A}r^k$, we apply a simple convolution operation to extract feature vectors from this leftover subgraph as follows:

$$x^k = \bar{W}^k \circledast \bar{A}r^k + b \tag{6.4}$$

We will have m extracted feature vectors as: x_1^k, x_2^k, ..., x_m^k, and $m = \binom{N}{K}$ if we consider all possibilities. However, the complexity of this operation will be $O(\binom{n}{k})$, which is unacceptable

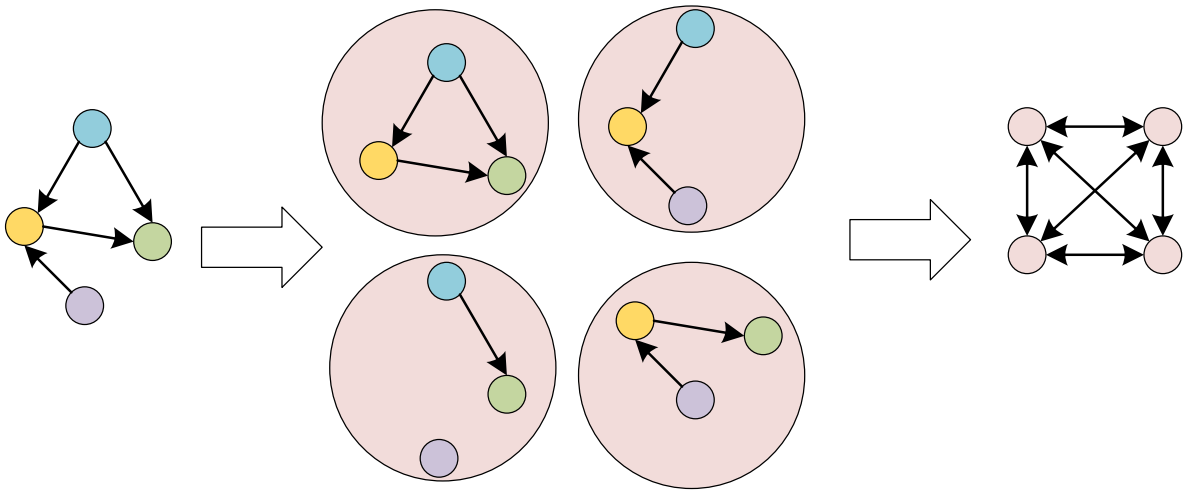


Figure 6.4: Convolution Kernel Example: Given input graph having 4 vertices, a 3-by-3 convolution kernel is applied. 4 convolution candidates are generated and the convolution results a new graph with 4 vertices

in practice. To relax it, in this chapter, instead of considering all possibilities, we will only pick s number of $\bar{A}r^k$ as convolution candidate which reduces the complexity to $O(s)$. Pick s samples of $\bar{A}r^k$ is a pooling/down-sampling operation which we will explain the details in Section 6.5. Finally, we consider the extracted feature vectors x_1^k, x_2^k, ..., x_s^k as a new feature graph $\mathcal{G}'$ with s number of vertices, and x_i^k is the feature vector for node i. An example is presented in Figure 6.4.

$\mathcal{G}'$ can be used as input to another Subgraph Convolution Layer in the proposed SGCNN architecture to form deep SGCNN model. However, notice that for $\mathcal{G}'$, we need to recalculate the adjacency matrix as well as the attribute matrix. Algorithm 10 summaries our Graph Convolution Kernel.

6.5 Graph Pooling Algorithm

In order to form deep structures for SGCNN, we propose a pooling operation before the convolution operations of each layer. The necessity of a pooling operation is because the combination of all possible convolution candidates is huge which has already explained in pre-

Algorithm 10: Graph Convolution Kernel.

Input: An input graph: G with n vertices
Input: An convolution kernel: $\bar{W}^k$
Input: Sample size: s
Output: An output feature graph: G'

1 Generate adjacency matrix $\bar{A}$ from G
2 Using the same vertices order to generate list of features $\mathcal{X}$
3 Create feature matrix $\bar{X}$ with n rows, and each row being $\mathcal{X}$
4 $\bar{A}r = \bar{X} \circ (\bar{A} + \bar{I})$
5 $m = \binom{n}{k}$
6 $CombList$ = Enumerating choice of $n - k$ elements from $1, 2, ..., n$
7 **foreach** $comb \in CombList$ **do**
8 $\bar{A}r_{comb}$ = remove rows and columns list in $cand$ from $\bar{A}r$
9 Add $\bar{A}r_{comb}$ into $CandList$
10 **if** $m > s$ **then**
11 Down-sample $CandList$ to s elements
12 **else if** $m < s$ **then**
13 Pad $CandList$ to s elements
14 **foreach** $cand \in CandList$ **do**
15 $x^k = \bar{W}^k \circledast cand + b$
16 Add new vertex v_k into G'
17 Add feature vector x^k on v_k
18 Connect v_k based on $cand$'s connection in G
19 **return** G'

vious section. Thus, with the a pooling operations inserted before convolution, the structure of our proposed subgraph convolution layer is presented in Figure 6.5

A nature idea of good downsampling/pooling operation is to remove the samples with less significant features. Thus, we consider to remove the subgraphs that has less total degrees compared to other samples for graph structured data. Algorithm 11 describes our proposed downsampling/pooling algorithm for graph.

In Line 1, we generate the adjacency matrix $\bar{A}$ from the input graph G. Lines 2-9 compute

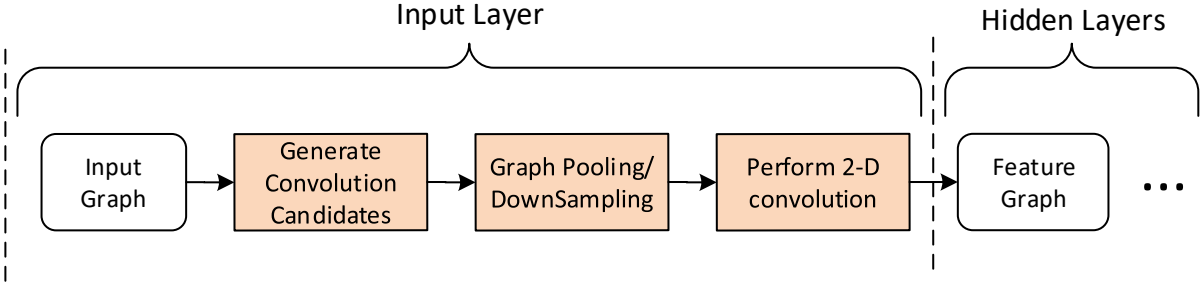


Figure 6.5: Subgraph Convolution Layer Architecture

Algorithm 11: Graph Pooling Algorithm.

Input: An input graph: G
Input: The list of the candidate nodes combinations: $Comb$
Input: Sample size: s
Output: The list of downsampled canndidate nodes: $Comb'$

```
1  Generate adjacency matrix Ā from G
2  foreach c ∈ Comb do
3      d_c = 0
4      foreach n ∈ c do
5          Calculate Degree of n as d_n
6          d_c = d_c + d_n
7      foreach c' ∈ Comb do
8          if c' is connected with c in Ā then
9              d_c = d_c + 1
10 Keep the s number of nodes with highest degrees and store in Comb'
11 return Comb';
```

the total degrees of each candidate nodes combination $Comb_l$, which are combinations of the nodes in G that are generated by the convolution kernel and these combinations will serve as the new nodes of the output feature graph after the graph convolution kernel. Specifically, lines 2-6 computes the total degrees inside the combination, and lines 7-9 computes the degrees in between different combinations. Finally, we will keep the s number of nodes combinations which have the highest degree.

6.6 Experimental Results

To evaluate the proposed non-euclidean training method, we applied our method on a dataset extracted from some samples in GrabCAD. GrabCAD is a free cloud-based collaboration solution that helps engineering teams manage, view and share CAD files [70]. GrabCAD consists of over 2 million open source engineering models, and we took 6090 models as training samples and 1523 models as testing samples for experimental purpose. From the GrabCAD website, we extracted the information (including model's name, model's author, model's parts names, model's tags, and model's comments, etc.) from the models and constructed graph from them. Notice that, the extracted information contains both social network data

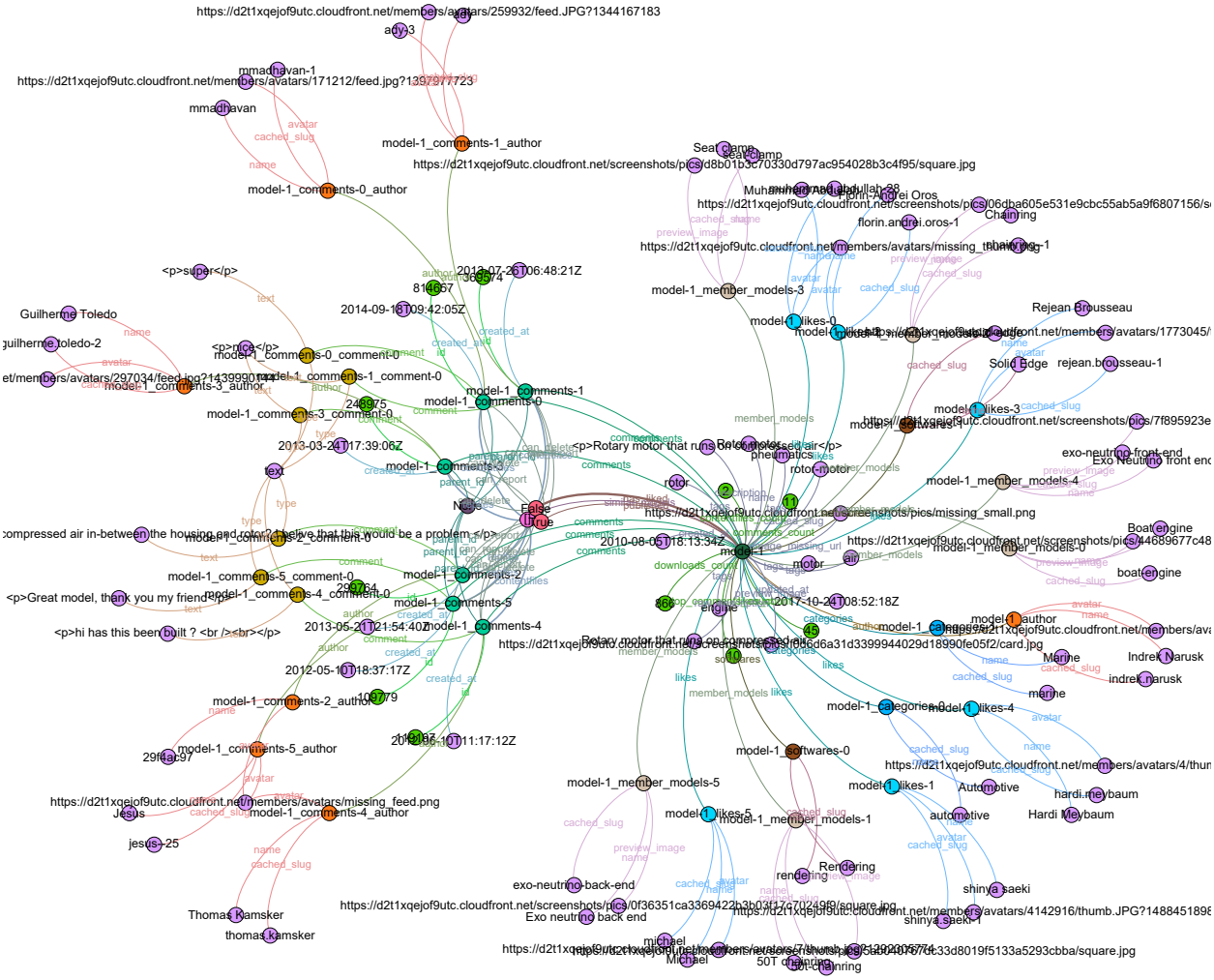


Figure 6.6: Example of a subgraph constructed from GrabCAD model

and engineering data. Nevertheless, we can use these data to construct graph and evaluate our method. One example of the constructed graph for one model is shown in Figure 6.6.

We classify all the samples into 4 different categories labeled as: (1) Car: 2237 samples; (2) Engine: 1623 samples; (3) Gear: 1731 samples ; (4) Robotic Arm: 2022 samples. A two layers SGCNN model is created for this learning task. The first layer has a kernel size of 16 by 16 and second layer has a kernel size of 4 by 4. The training results are shown in Figure 6.7 and Figure 6.8. The x-axis is the number of training echoes and the y-axis is the loss value and accuracy respectively.

The results show that, with the current dataset, we could converge to an accuracy of around

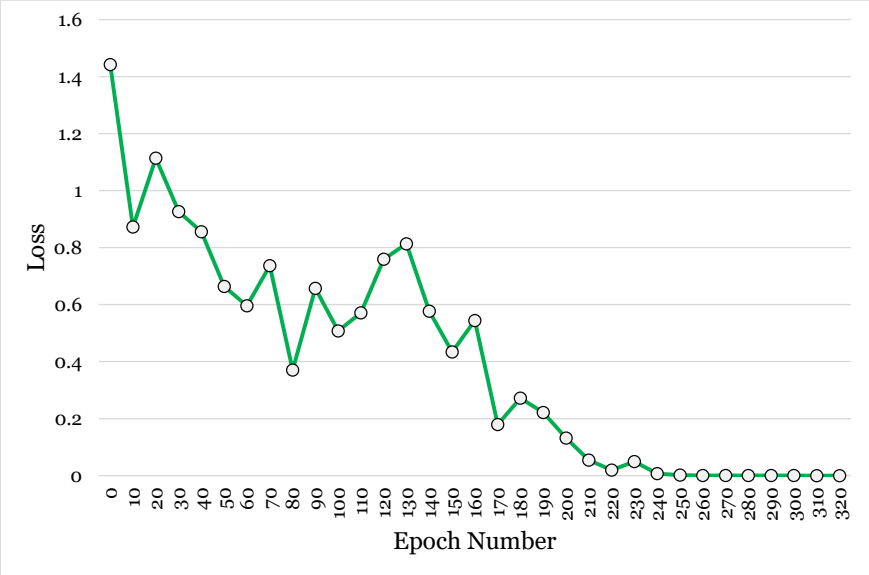


Figure 6.7: Loss value during the training process

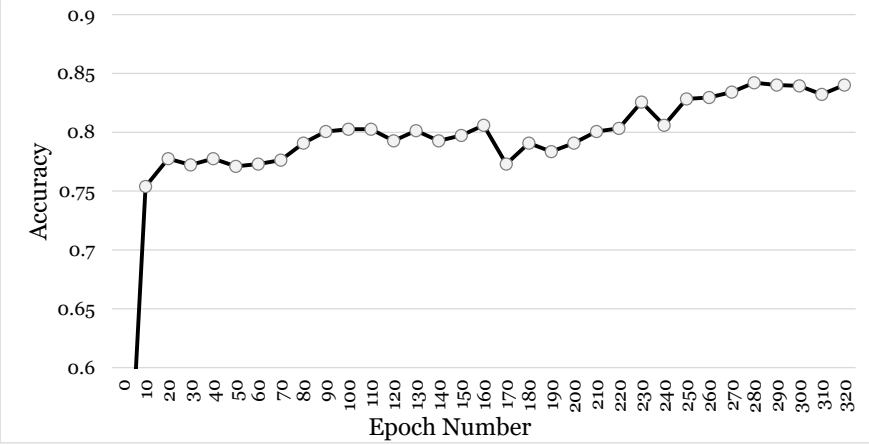


Figure 6.8: Accuracy results during the training process

85%. This preliminary test results demonstrate that we can train machines to learn functionality of a engineering system using proposed non-euclidean training algorithms. Further testing with more engineering data from different sources will be conducted in future.

6.7 Conclusion

This chapter proposes a novel non-euclidean training method for engineering data. A Structured Graph Convolutional Neural Network (SGCNN) is developed to perform graph invariant learning tasks at graph and subgraph level. The proposed method enables the design automation of CPS at system level with the help of machine learning technology. The preliminary results show that we can achieve an accuracy of around 85% for machines to learn the functionality of a CPS.

Chapter 7

Security-Aware Functional Modeling of Cyber-Physical Systems

7.1 Introduction and Related Work

The increasing amount of software and communication networks in Cyber-Physical Systems (CPS) deployed in many industries is also making them more vulnerable to the cyber-attacks [91, 141]. One easy-to-relate example is automobile design; over the years it has shifted from being mechanical-centric to software-centric [35]. It has been shown that different types of attackers and attack scenarios such as through wireless access to the car, through physical access to a car, and through access to the in-vehicle network (tampering Electronic Controls Units (ECU)) may happen in the real world scenarios [40, 94, 149, 165]. An important observation is that even though systems are becoming software-centric, attacks can be physics-based. In [158], the authors have implemented an ABS spoofer that could physically attack on the wheel speed sensor so that the functionality of the Anti-lock Braking System (ABS) is completely distorted. These principles can be applied to other

complex systems that control critical infrastructure.

Unfortunately, there is currently a gap between the design automation tools and cybersecurity. This chapter advocates the development of new methodologies and tools to support designers, as early as possible during the design cycle, to identify and solve cybersecurity problems in complex systems [96]. In [182], a Model-Based Design (MBD) method to assess the security of CPS is proposed with four architecture-level attack models. Authors in [128] have discussed a MBD technique to quantify the security metrics at the early design stage. To achieve the proposed metrics, a security modeling method is proposed in this chapter [128]. Furthermore, graph-based modeling methods are used in many security problems. Authors in [190] have proposed a systematic method for analyzing cyber-attacks on CPS based on an extended Data Flow Diagram (xDFD) approach. Existing work in [126] has proposed an attack tree based approach for the system-level security design. However, most of the modeling methods used in the existing works are limited on modeling the cyber domain software for security analysis.

At the early design stage, MBD is widely used for CPS design. Our work proposes the formulation of the security problem or attack patterns *before* the system is built. We exploit the same observation that in concept design where identifying and fixing problems at the early stages is economically beneficial for the costs, performance, and reliability of the system. Therefore, we have developed a design automation tool that uses simulation to validate cybersecurity vulnerabilities at the system-level. We model cybersecurity attacks and countermeasure functionalities using a novel *security-aware functional modeling* language (see Section 7.3) implemented in the commercial design and simulation tools (see Section 7.4). Currently, our library of attacks consists of six attack models. We validate the methodology and demonstrate that our idea can be integrated to current design workflows (see Section 8.7). Our experimental results show the benefit of our methodology using an automotive use-case.

7.2 Our Novel Contributions

Building upon our previous work on functional modeling [32, 33, 35, 179], this chapter proposes a *security-aware* functional modeling methodology. Moreover, since both the cyber domain (control blocks and signal flows) and the physical domain (physical process and energy flows) may be modeled using functions, we extend this notion to represent *cybersecurity functions* that represent attacks and countermeasures. The benefit of such methodology, is that we characterize the cyber-physical security problem in an implementation independent manner.

The novel contributions of this chapter are:

1. A security-aware functional modeling methodology.
2. Cybersecurity functions and their associated attack and countermeasure models.
3. System-level modeling and simulation of attacks beneficial for the concept design stage of CPS.

7.3 Security-Aware Functional Model

Traditional functional models describe only two types of functions: physical and cyber. Functions interact with each other through flows that reflect the base functions using energy, material, and signal flows. These flows carry real physical and cyber properties such as mechanical, electrical, thermal energy, and data. Thus, existing functional models naturally *leak* information that can be used to attack the system via the signal flows in the cyber domain or energy/material flows in the physical domain. We extend the functional modeling concept and include cybersecurity functions. Our proposed *security-aware functional models*

provide the means to both analyze the effect of cybersecurity attack functions in the system, and refine the design using cybersecurity countermeasure functions.

An example of a security-aware functional model for a car is shown in Figure 7.1. In this example, both a cyber and a physical attacks are modeled. The blue arrows in the flows represent the cyber attack vectors, and the red arrows represent the physical attack vectors. The purpose of the analysis is to quantify the effects of these attacks in the `Export TME` function that maps to the velocity of the car. In other words, "is the speed control of the car vulnerable to a cyber-physical attack?".

In order to evaluate the security level of the Export TME function, we simulate the model and analyze the impact of the attack at the system-level. This step is important to identify functions with low security levels. These low security functions can be protected by refining the functional model and adding cybersecurity countermeasure functions. This iterative approach facilitates the analysis of different scenarios. An important benefit of the iterative approach is that more complex scenarios can be composed.

From the analysis of this security-aware functional model, a system-level simulation model can be directly generated using the synthesis technology provided in the existing works [32, 33, 35, 179].

7.4 Design Automation Tool

To validate the methodology, we have implemented a design automation tool using commercial of-the-shelf software as shown in Figure 7.2. Modeling of the system functions (multi-physics), cybersecurity functions, and scenarios (e.g., environmental conditions) is done in Amesim in order to take advantage of the libraries of subsystems. And the modeling of attack models is done in Matlab/Simulink because we take advantage of the advanced

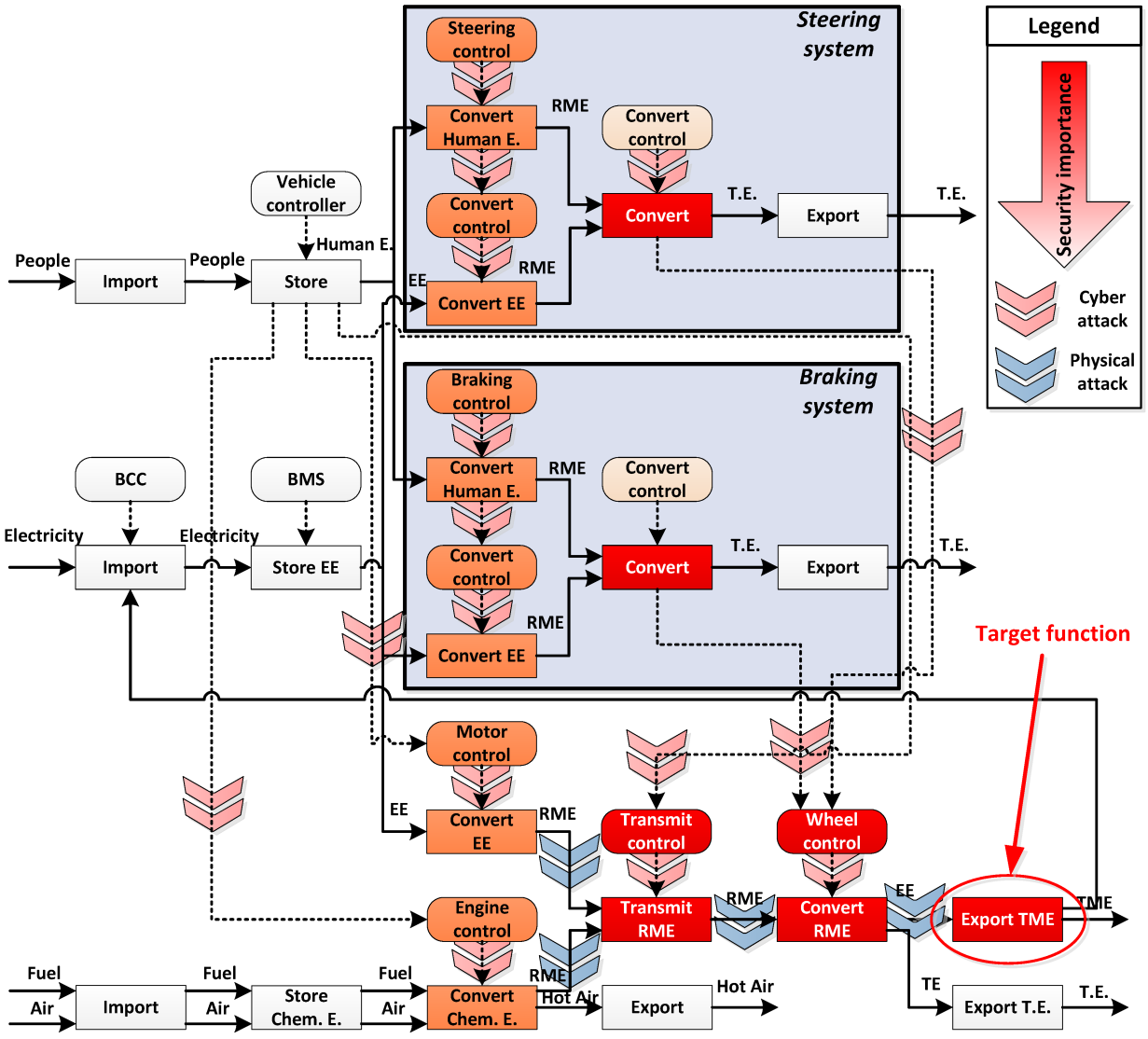


Figure 7.1: Security-aware functional level model of automotive.

mathematical capabilities of the software.

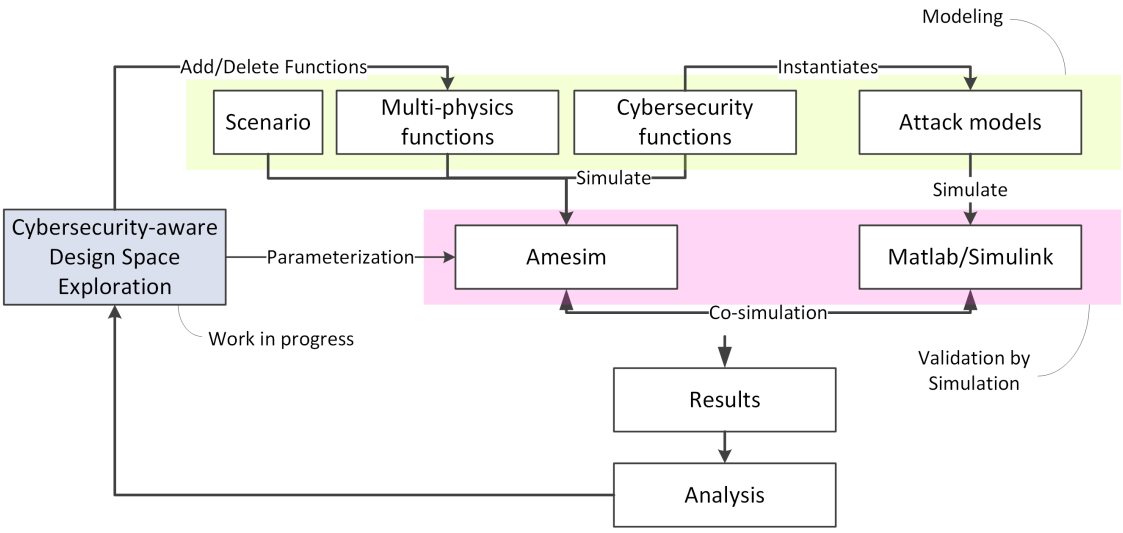


Figure 7.2: Our proposed design automation tool.

Notice that cybersecurity functions in Amesim are proxies for the attack models modeled in Matlab/Simulink; this is shown in the screenshot of our tool in Figure 7.3. Cybersecurity functions are instantiated into attack models during simulation. The composed system is validated by simulation using a co-simulation between Amesim and Matlab/Simulink. All the functions are instantiated to components in Amesim libraries, and the cybersecurity functions are instantiated in the Matlab/Simulink attack models. The results are analyzed by the user using standard time series and plotting capabilities of the simulation software. Currently, we are working on automating the analysis process and performing a cybersecurity-aware design space exploration. Our goal is to allow the tuning of parameters in the system to make it more resilient to attacks in different scenarios.

We have developed and integrated six attack models to our design automation tool. The developed attack models are mathematically represented as follows. u_s is the original signal/flow and v_s is the attacked signal/flow. Most of the attack happens during the attack period from τ_{start} to τ_{end}.

Fuzzy attack model [182] adds a random distribution (e.g., uniform distribution $unif(atk_{upper}, atk_{lower})$

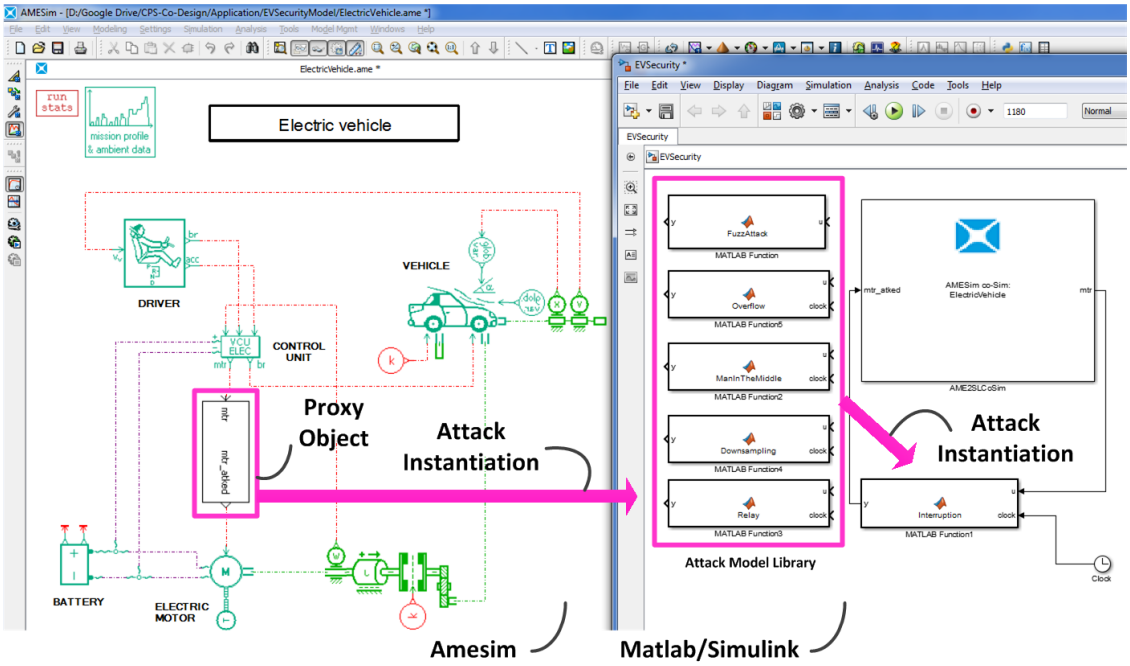


Figure 7.3: Implementation of the proposed design automation tool.

triggered by the poisson clock $fclock_{pois}$) to the selected functional signals/flow to destabilize the system.

$$
\begin{aligned}
&Event = fclock_{pois}(\lambda) \\
&atk_s = \begin{cases} unif(atk_{upper}, atk_{lower}) & Event \neq 0 \\ atk_s & Event = 0 \end{cases} \\
&v_s = u_s + atk_s
\end{aligned} \tag{7.1}
$$

Interruption attack model [182], also referred to as denial-of-service attack, stops the

functional signal/flow during the attack period.

$$v_s = \begin{cases} 0 & \tau_{start} < \tau < \tau_{end} \\ u_s & else \end{cases} \tag{7.2}$$

Man-in-the-middle attack model [182] mimics a human attack behavior. When the attack occurs, the functional signal/flow is changed to the manipulated signal/flow controlled by the attacker.

$$v_s = \begin{cases} u_m & \tau_{start} < \tau < \tau_{end} \\ u_s & else \end{cases} \tag{7.3}$$

Replay attack model [182] records the functional/signal flows over a period of time in a vector. When the attack starts, this attack model replaces the actual signal/flow with recorded data of other subsystems.

$$u_r[\tau \bmod \tau_{period}] = u_s$$
$$v_s = \begin{cases} u_r[\tau - \tau_{start}] & \tau_{start} < \tau < \tau_{end} \\ u_s & else \end{cases} \tag{7.4}$$

Overflow attack model [40, 116] extends the number of bits of the message in a signal

in order to overflow the buffers on the receiver's side.

$$v_s = \begin{cases} u_s >> Num_{bits} << Num_{bits} & \tau_{start} < \tau < \tau_{end} \\ u_s & else \end{cases} \tag{7.5}$$

Down-sampling attack model [21] reduces the sampling rate of functional signal/flow. This attack is known to reduce the Quality of Control (QoC) of a system.

$$u_{low} = \begin{cases} u_s & \tau \bmod rate_{down} = 0 \\ u_low & else \end{cases} \tag{7.6}$$

$$v_s = u_{low}$$

7.5 Experimental Results

To demonstrate the proposed design automation tool and the attack models during the early design stage, we use the design of an Electric Vehicle (EV) as a case study shown in Figure 7.3. First, we run the simulation of the ELECTRIC MOTOR without instantiating any attack model, the results are shown in Figure 7.4. In order to evaluate our attack models on a realistic scenario, we inserted the attack model interface in between the motor and the ECU controlling the motor. The attack model interface is instantiated by one of the attack models from the attack model library as presented in Figure 7.3.

Figure 7.5(a) shows that the fuzzy attack may destabilize the motor by adding noise to the control signals. Figure 7.5(b) shows that the control signal is interrupted and out of service

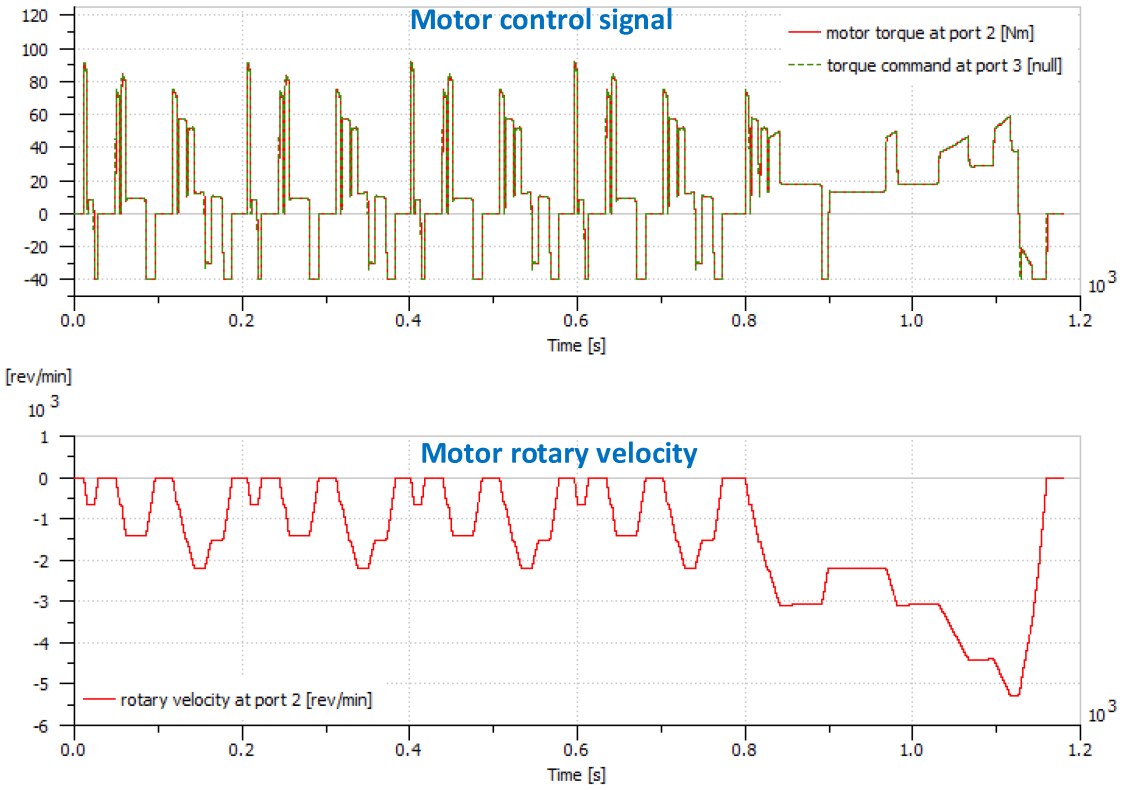


Figure 7.4: Simulation results without attack models.

from 200s to 800s. Figure 7.5(c) shows that the motor is controlled by a linear signal that is defined manually by the attackers in the attack model. Figure 7.5(d) shows that, instead of getting control signals based on the current status, the attack model is replacing the signals with recorded signals from 200s to 300s. Figure 7.5(e) is showing that the overflow attack reduces the accuracy of the control signals by overflowing the buffer, thus the lower-bits of the control signals is lost. As a result, the total Quality of Control (QoC) is reduced. However, in this example, the simulation results demonstrate very few reductions on QoC, which means the overflow attack will not affect this signal. Figure 7.5(f) is showing that the down-sampling attack model will reduce accuracy of the signals, and thus reduce the QoC of the motor.

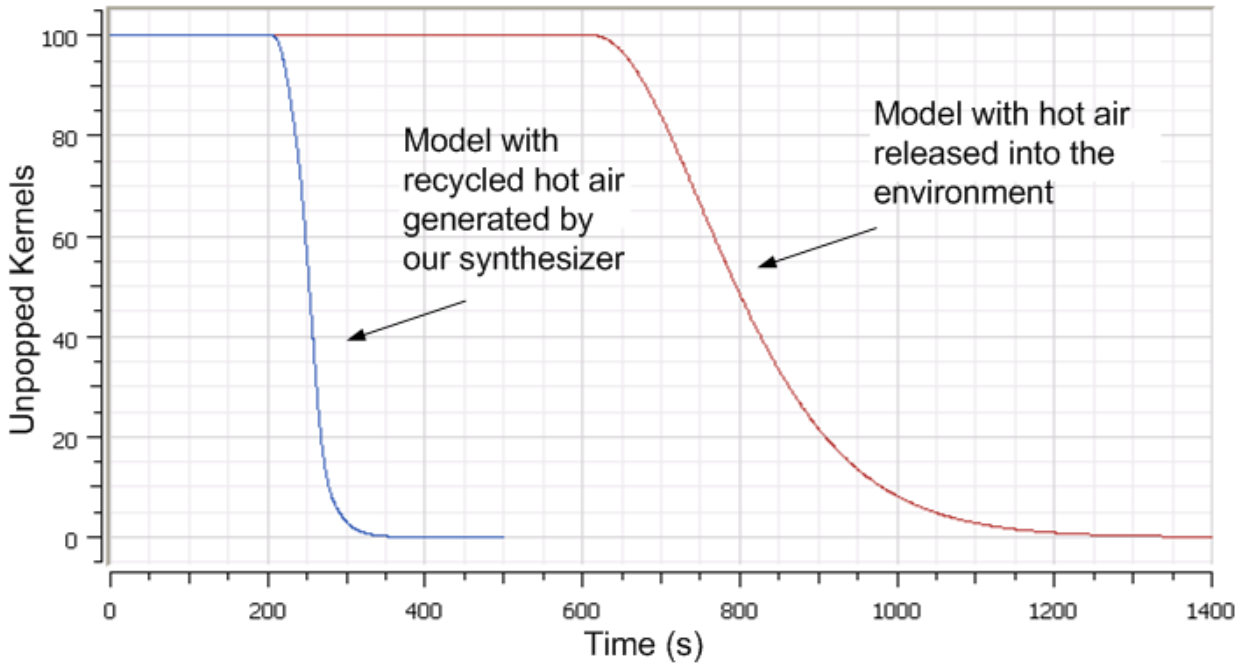


Figure 7.5: Simulation results with the proposed attack models: (a) fuzzy attack, (b) interruption attack, (c) man-in-the-middle attack, (d) replay attack, (e) overflow attack, (f) down-sampling attack.

7.6 Conclusion and Future Work

In this work, functional model is used to help the analysis of security problem during the early design stage. In future, we will explore the capability of the functional model to not only help the analysis, but also to automatically generate/synthesize system-level simulation models including attack models and system architecture to help the evaluation of security strength. Moreover, for the demonstration purpose, six different attack models are developed in the current attack library. Additional attack models which may capture both cyber domain and physical domain attacks will be developed to enrich the library in our future work. Moreover, to help the system designers to quickly identify the security problem during the early stage. Security cost metrics may be developed and used in the proposed security-aware functional models.

Chapter 8

Securing Wireless Communication in Automotive CPS

8.1 Introduction

Wireless technologies are widely implemented in automotive Cyber-Physical Systems (CPS) for navigation schemes (e.g., GPS) and infotainment applications such as hands-free calling, and satellite radio [51]. As a light-weight solution to the wireless harnessing problem [107] and for its aforementioned applications, wireless technologies applied on many microcontrollers all throughout a vehicle can enable a powerful improvement in safety and comfort for people and functionality and efficiency for automotive CPS [50, 56]. A notable example is the Tire Pressure Monitoring System (TPMS), which is implemented in many modern vehicles and utilizes several controllers and tire sensors to measure and display tire temperatures and pressures to passengers. Through the TPMS, passengers can understand from warning signals when to re-inflate or replace their tires, leading to evasion of unnecessary dangers.

An astounding 80% of all vehicular[1] collisions is caused by drivers but it is clear that wireless technologies can greatly reduce the risk of driver-caused collisions and improve traffic efficiency [183]. In order to realize these objectives, federal agencies (e.g., United States Department of Transportation) and research organizations (e.g., Google) are developing general wireless vehicular communication protocols (V2X), which we define as either Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) [76]. With V2X communication, vehicles can perform collision detection and prevention, path control, traffic management, environmental hazard avoidance, and new forms of entertainment through Internet connection. Below, Figure 8.1 provides an illustration of a setting with V2X communication.

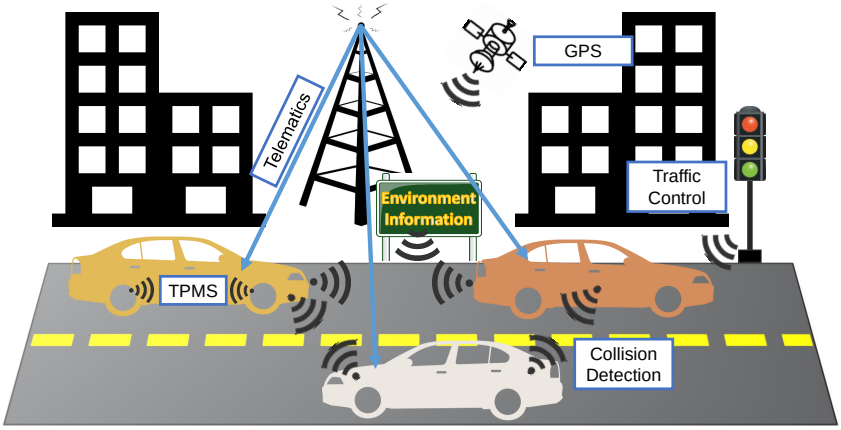


Figure 8.1: Examples of V2V and V2I Applications

8.1.1 Motivation

With a new model of interconnectivity, both old and new V2X applications will connect traditionally isolated vehicles to each other, infrastructure, satellites and other entities through insecure wireless channels. Already, security concerns have arisen over leakage of critical information about the vehicle or the passengers and over the possibility of indirect control of the vehicle's mechanisms [41, 101, 121, 189]. In fact, these security concerns date back to the mid-1990s, a time where many vehicles used Remote Keyless Entry (RKE). Researchers

[1]The contents of our proposed techniques can apply to all types of automotive cyber-physical systems, however we use the average vehicle (e.g., car, truck, or any other thing used to transport people or goods) as a motivating example.

eventually found that RKE was vulnerable to relay attacks, where relaying a signal through simple technology could unlock and start a vehicle when its owner is away [19]. In 2005, a Texas Instrument RFID transponder implemented in millions of vehicles was found to be hackable, thus portraying another security threat [19]. Then in 2010, researchers developed an attack that captured and read TPMS communication packets from a vehicle up to 40 meters (.02 miles) away. Furthermore, they demonstrated the possibility of injecting packets into the TPMS network to trigger a fake warning signal [84]. In the recent years of 2014 and 2015, researchers with support from the Defense Advanced Research Projects Agency (DARPA), have developed and demonstrated exploitable hacks regarding vehicular infotainment applications and systems like UConnect [121]. Their hacking demonstrations ended in the recall of many vulnerable vehicles, such as Chrysler [122]. An introduction of V2X communication will eventually cause similar and new security concerns. As a result, researchers such as those from the European Telecommunications Standards Institute (ETSI) are proposing the following security objectives for V2X communication: confidentiality, integrity, availability, accountability and authenticity (for more details, please see the technical reports [58]).

8.1.2 Security Challenges for V2X Communication

For the purposes of our work, we focus on the security requirement of confidentiality. We summarize that for wireless communication in automotive CPS, messages will need to be encrypted depending on the confidentiality requirements of applications [154]. As a simple example, account information will need to be encrypted in financial applications like Electronic Toll Collection (ETC) [101, 138] and for cooperative pre-crash sensing [45]. Another major challenge for V2X communication is authentication. Since users are exposed to many dangers due to the wireless communication, it is necessary for a receiver to verify that a transmitted message was generated by a legitimate user. Recently, researchers

have proposed to solve the authentication problem in automotive communication in an Ad-Hoc manner [43, 106]. However, this type of scheme requires an established secure channel for exchanging authentication information such as secret keys and identifications before communication. It is important to note that these and other security objectives apply to resource-limited (in terms of computational power, energy consumption and memory size) time-critical embedded devices (e.g., sensors, V2X) and resource-limited non-time-critical devices (e.g., infotainment) within the vehicles. Because of their important role in keeping passengers and drivers safe, we center our attention on the resource-limited and time-critical devices.

A typical automotive design needs to provide security for about 20 years or more [154, 178], implying the necessity of a reliable and efficient cryptographic scheme to achieve some of the aforementioned security objectives. Cryptographic algorithms fall under two categories: **1)** Symmetric and **2)** Asymmetric. As seen in Table 8.1, symmetric algorithms, such as the Advanced Encryption Standard (AES), have very high performance and lower energy overhead [135] in comparison to asymmetric algorithms, such as RSA and Elliptic Curve Cryptography (ECC). However, both of these schemes are challenging to implement on resource-limited and time-critical devices.

The major problem of using a symmetric encryption algorithm is that both parties must share a secret key to establish a secure communication. On the other hand, although asymmetric algorithms do not require a shared secret key for secure communication, they are too slow for the majority of time-critical applications and too resource-intensive in terms of computational power and memory usage [127, 135, 154]. As a result, higher performance processors have been used to address these issues. However, using such processors (e.g., Qualcomm Snapdragon 602A for V2X) comes with a non-negligible cost (for example, the Qualcomm Snapdragon 602A may involve around $1000 or more in extra cost). Moreover, there are up to 100 Electronic Control Units (ECUs) in a modern car, and many of these ECUs are

low cost micro-processors. An alternative security method would be necessary to enable V2X applications on these processors. In some of the state-of-the-art approaches, research groups and government organizations have proposed the use of hybrid solutions to reduce overhead from the asymmetric algorithms [154, 155]. In a hybrid solution, a symmetric key is generated from a Pseudo Random Number Generator (PRNG) or a Key Encapsulation Mechanism (KEM) [82] and exchanged through an asymmetric algorithm. Afterward, higher performance can be achieved through symmetric encryption of data.

Table 8.1: Comparison of Existing Cryptographic Algorithms

	Symmetric	Asymmetric	Hybrid
Authentication	Message Authentication Code (MAC)	Digital signature	Digital signature on keys MAC on data
Confidentiality	Encryption of data	Encryption of small data	Encrypt keys with Asym. Encrypt of data with Sym.
Performance	Very fast	Slow	Medium
Code size	Thousands of bytes	Thousands of bytes	Thousands of bytes
Key size	32-256 bits	ECC: 256-384 bits RSA: 1024-3072 bits	512-3072 bits for Asym. 32-256 bits for Sym.
Key management	Random key generation Pre-shared secret key	None	Random key generation

However, there are still three major limitations to the current hybrid approach: **1)** It requires a key exchange session which uses an asymmetric algorithm whose lengthy computation time is generally not acceptable for safety-related applications which require a reaction time of 50 to 200 milliseconds [154]. **2)** The hybrid solution requires an implementation of the asymmetric algorithm in the embedded devices, thus causing non-negligible memory space overhead. **3)** Similar to symmetric algorithms, the hybrid solution generally relies on a Pseudo Random Number Generator (PRNG) or user-given inputs to help produce a symmetric key with high entropy. These approaches, however, cannot provide enough entropy[2] due to high levels of predictability of the seed or user-given inputs and deterministic nature of the key generation algorithm [129]. **For the aforementioned reasons, secret key generation and exchange are considered challenging problems for automotive**

[2]Entropy (more specifically Shannon Entropy) can be used as a quantified value of randomness for a set of bits.

wireless applications.

8.1.3 Related Works

To help solve this problem of developing a reliable yet efficient and fast encryption mechanism, researchers have been looking toward physical randomness as a high entropy source. As an example, researchers proposed the use of physical randomness in circuit characteristics to generate secret keys [144, 166]. Similarly, it is possible to exploit the physical randomness from wireless communication channel characteristics, such as the multipath-induced fading and shadow fading to generate strong secret keys. Most of the state-of-the-art theories and practical methods for generating secret keys using physical characteristics of the wireless channel (or the physical layer) have been proposed within just the last decade but have not been applied to the automotive CPS environment [18, 85, 111, 133, 142, 181, 191, 192, 194].

The success of generating secret keys based on the wireless channel's physical properties depends on three properties: **1)** Reciprocity of the radio wave propagation, **2)** Temporal variation, and **3)** Spatial variation in the wireless environment (see details in Section 8.2). Besides most of the theoretical works [18, 111], some practical implementations for sensor network applications [12, 85, 142] have been performed and rely on the Multiple-Input and Multiple-Output (MIMO) approach or collaborations among multiple wireless nodes to create secret keys with high entropy. Work in [192] has also provided an implementation for V2X applications. However, it mainly focuses on comparing their algorithm with other key generation algorithms and modeling the spatial and temporal variations of the automotive wireless channel. Moreover, the authors do not consider practical challenges such as abiding by real-time requirements for safety-critical V2V applications and optimizing their algorithm to its highest potential in terms of resources such as time, energy, and memory. Lastly, other types of physical layer security have been discussed in [127] but have not yet been applied

to the automotive CPS domain to enable efficient and reliable security protocols as we have done.

In summary, solving the limitations of the above-mentioned state-of-the-art approaches to secure wireless communication in automotive CPS poses the following challenges:

1. **Finding a reliable high entropy source** to generate secret keys for symmetric cryptographic algorithms.
2. **Designing a reliable solution** for the management of symmetric secret keys.
3. **Optimization of the solution and key size** in terms of performance.

8.1.4 Overview of Contributions

To address the challenges, we propose a novel secured communication scheme for automotive wireless communication. The foundation of the proposed scheme is the technique from our published paper in the International Conference on Cyber-Physical Systems [180], which generates symmetric cryptographic keys based on a **pre-shared key (PSK)**[3] from the physical randomness of the automotive wireless channel under tight memory and performance budgets. The biggest advantage of this technique is that it solves the key generation and exchange problem at the same time, which means it can generate the PSKs with high entropy while eliminating the costly requirements of the asymmetric algorithms for the key exchange process. Moreover, it can also replace the asymmetric algorithm for exchanging secret keys and identification for authentication purposes. In comparison to our previous paper, the novel contributions of the proposed scheme include: 1) Protocol algorithms to support the proposed PSK generation technique: including Scenario Mapping, PSK Optimization, and

[3]A PSK is also known as a shared secret (a random bit string known only to a pair of communicating parties).

Cryptographic Key Derivation; 2) New simulation results corresponding to these algorithms; and 3) New experimental results, including more detailed comparisons between our technique and RSA and ECC. **To the best of our knowledge we are the first to demonstrate, through realistic automotive modeling, simulation and experiments, that higher levels of entropy and performance may be obtained from the moving and changing environment to practically generate symmetric secret keys for automotive CPS wireless communication.** The contributions of this chapter are as follows:

1. **Automotive wireless communication system models (Section 8.2)** including the channel and attack models from a security perspective.
2. **A secured communication scheme for automotive wireless communication** which includes:
   (a) **A wireless channel-based PSK generation technique (Section 8.3)** that incorporates the results from optimization to generate a PSK for two communicating parties in automotive communication.
   (b) **A PSK length optimization technique (Section 8.4)** under timing constraints based on the scenario of vehicular communication.
   (c) **Cryptographic key derivation methods (Section 8.5)** to convert the PSK into a suitable key for encryption purposes.
3. **Simulations and real world experiments (Sections 8.6 and 8.7)** to demonstrate the practicality of our proposed PSK optimization and generation techniques.

For a full visual description of our secured automotive wireless communication scheme, please see Figure 8.2.

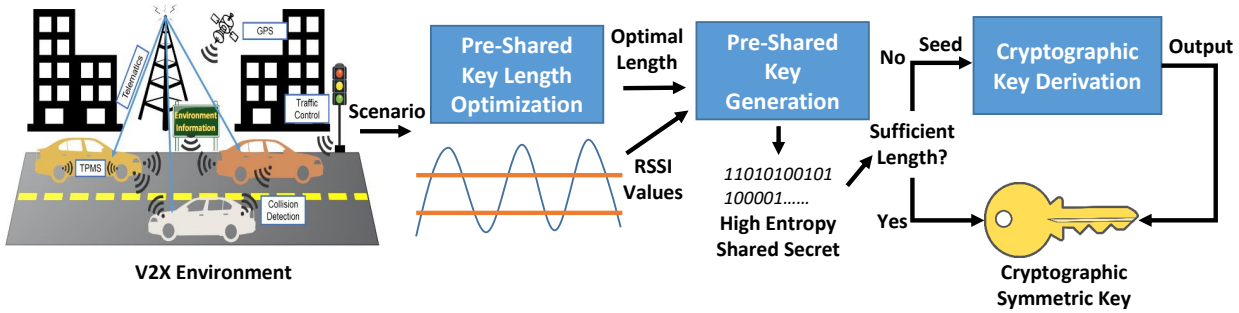


Figure 8.2: Overview of the Proposed Scheme for Secured Automotive Wireless Communication

8.2 System Model

8.2.1 Wireless Communication Model

We provide a sender-to-receiver model of an automotive wireless communication system, where an ECU or sensor-node inside a vehicle A is communicating with a device from another vehicle or infrastructure B in the presence of an eavesdropper in vehicle E. In this model, the sending signal S_A from A over the wireless channel will be received by B and E as follows:

$$\begin{aligned} R_{A\to B}(t) &= H_{A\to B}(t) \times S_A(t) + N_{A\to B}(t); \\ R_{A\to E}(t) &= H_{A\to E}(t) \times S_A(t) + N_{A\to E}(t); \end{aligned} \tag{8.1}$$

where H is the channel gain and N is the zero mean additive Gaussian noise [191]. If B responds with a signal R_B to A, then the received signals by A and E may be modeled as follows:

$$\begin{aligned} R_{B\to A}(t) &= H_{B\to A}(t) \times S_B(t) + N_{B\to A}(t); \\ R_{B\to E}(t) &= H_{B\to E}(t) \times S_B(t) + N_{B\to E}(t); \end{aligned} \tag{8.2}$$

Suppose, $S_A(t)$ and $S_B(t)$ are two probe signals, known to A, B, and E. From the received signals $R_{A\to B}(t)$, $R_{B\to A}(t)$, $R_{A\to E}(t)$, and $R_{B\to E}(t)$, the channel gains can be estimated and are denoted as $H'_{A\to B}(t)$, $H'_{B\to A}(t)$, $H'_{A\to E}(t)$, and $H'_{B\to E}(t)$, respectively. Due to the **reciprocity property** [191] of the wireless channel, if A and B send the probe signals to each other within the wireless channel's coherence time[4], we can assume that the estimated channel gain is the same for both A and B: $H'_{A\to B}(t) \approx H'_{B\to A}(t)$. However, from the eavesdropper's side, the estimated channel gains $H'_{A\to E}(t)$, $H'_{E\to A}(t)$, $H'_{B\to E}(t)$ and $H'_{E\to B}(t)$ will be independent of $H'_{A\to B}(t)$ and $H'_{B\to A}(t)$, if the eavesdropper is a few wavelengths [191] away from the legitimate wireless channel. Utilizing this concept, the channel gain ($H'_{A\to B}(t)$ and $H'_{B\to A}(t)$) may be used to extract PSK bits (see our technique in Section 8.3) for security purposes.

The wireless communication channel gain varies over time due to temporal or spatial variations in the environment. Typically, the channel may be modeled with a fast fading model or a slow fading model depending on the changing speed of the environment [159]. For automotive CPS, if there exists a velocity difference between two communicating automotive wireless nodes, we use a **fast fading model (temporal variation)**, otherwise, we use a **slow fading model (spatial variation)**.

Fast Fading

For the fast fading model, we assume a *Rayleigh fading channel* [159] which is suitable for modeling vehicular wireless communication [159] in an urban driving profile. The *Rayleigh fading channel* models the *Doppler shift effect* [159] due to the different speeds between two communicating wireless nodes. In this model, the channel gain H should abide by the

[4]In a wireless communication system, the coherence time is the time duration over which the channel impulse response is considered to be invariant.

following Probability Distribution Function (PDF):

$$PDF_H(H, \sigma) = \frac{H}{\sigma^2} e^{-H^2/(2\sigma^2)} \tag{8.3}$$

where σ is an environment-related parameter. Due to the *Doppler shift effect*, H only remains constant within the coherence time [159] T_c (see the following Equation).

$$T_c \approx \frac{0.423}{f_d} \tag{8.4}$$

here, f_d is the maximum *Doppler frequency*. During automotive wireless communication between A and B, f_d may be decided by the speed difference of the two communicating vehicles ΔV_A as shown below:

$$\begin{aligned} f_d &= \frac{\Delta V}{c} f_0 \\ \Delta V &= |V_A - V_B| \end{aligned} \tag{8.5}$$

where c is the speed of light and f_0 is the carrier frequency.

This model reflects that the channel changes roughly every time interval of T_c. In other words, the higher the ΔV is, the more frequent the channel changes and the quicker a channel-based PSK may be generated. Extracting information from the channel gain H to generate a secret bit must be done within a given time period, T_c. Otherwise, the changes in the channel after T_c may cause mismatches between the generated PSKs of the communicating automotive wireless nodes.

Slow Fading

When the relative speed between the communicating automotive wireless nodes is low, $\Delta V \approx 0$, the fast fading model will not work. Therefore, we use a general slow fading model for the wireless communication. In a slow fading channel, the gain remains correlated in time if the channel does not move over a certain distance. This distance is defined as the coherence length L_{coher}. On the other hand, the model assumes that if the channel moves further than L_{coher}, the channel gain will become independent of the previous channel gain. Therefore, considering the velocity of the vehicle V, we may calculate the coherence time for a slow fading channel as follows:

$$T_c \approx \frac{L_{coher}}{V} \tag{8.6}$$

Similar to the fast fading channel model, the slow fading channel also changes roughly every time interval, T_c. As demonstrated with Equation 8.6, L_{coher} is decided by the environment. In other words, the higher the V is, the more frequently the channel physically changes. The time varying channel gain for a slow fading model follows the log-normal distribution as shown below:

$$PDF_H(H, \sigma) = \frac{1}{H\sigma\sqrt{2\pi}} e^{-\frac{ln(H)}{2\sigma^2}} \tag{8.7}$$

8.2.2 Attack Model

In this chapter, we consider **a classic non-intrusive wireless attack model** where the attacker tries to decipher the message by eavesdropping on packets from the legitimate wireless channel through a separate wireless channel. We assume that the attacker can capture all the wireless packets sent through the legitimate wireless channel and the attacker

knows all the information about the communication system including modulation/coding techniques and cryptographic algorithms. Therefore, in such a scenario, if the attacker can get the related pre-shared or cryptographic key, the system security requirements will be broken. As a result, we define attack strength *AttackStr* as a rate at which the attacker can employ a given amount of computing hardware resources to evaluate a number of keys within a period of time. We note that intrusive attacks are not considered in this chapter since they typically require the use of highly expensive and impractical devices and are challenging to implement on specific vehicles in real-time scenarios. Further, we note that no knowledge about the attacker is necessary (such as channel state information [127]) for our algorithm as the attacker will generally be farther than a wavelength (approximately 5 centimeters for the 802.11p automotive communication protocol [162]) away from the legitimate wireless channel. Notice that, in the case of V2I communication, we also assume that the infrastructure is typically physically protected so that the attackers may not eavesdrop the messages within a few wavelengths from the infrastructure.

8.2.3 Security Strength Evaluation

Security strength indicates the amount of work (number of operations) an attacker would need to do in order to break a cryptographic algorithm or system. According to the National Institute of Standards and Technology (NIST) standard, an algorithm or system is defined to have "N-bits security strength" when it requires an attacker to perform around 2^N operations to break the algorithm or system [13]. In our case, the security strength is equivalent to the number of bits in the pre-shared key, which is the basis for the cryptographic key.

Nonetheless, in order to directly measure the randomness and security strength of the PSK, we choose to use the concept of min-entropy [83]. As a worst case estimation, min-entropy provides the lower bound of randomness. Let K be the set of all possible PSKs randomly

generated, the min-entropy is defined as follows:

$$H_{\infty} = H_{min} = -log(\max_{k \in K} Pr[K = k]) \tag{8.8}$$

where, $Pr[K = k]$ is the probability of generating PSK $k \in K$.

Thus, we model the security strength, $Security_{str}$, of a cryptographic algorithm or system using the average min-entropy on each bit of the key as follows:

$$Security_{str} = H_{min}/Key_{size} \tag{8.9}$$

where, Key_{size} is the size of the key and $Security_{str}$ is a value ranged from 0 *to* 1 in the unit of bits. For example, a 128-bit key with $Security_{str} = 0.5\ bit$ will have 64 *bits* of min-entropy.

8.3 Pre-Shared Key Generation

We present our PSK generation algorithm with a V2V wireless communication example shown in Figure 8.3. In this example, both Alice and Bob are driving, where Alice's vehicle (A) is communicating with Bob's vehicle (B). Assume the driving velocities for A and B is V_A and V_B, respectively and the velocity difference between these two moving vehicles is ΔV. The coherence time T_c of the communication channel between A and B may be estimated using Equation 8.4 and Equation 8.5. Now, if A and B want to generate a PSK with size of PSK_{size}, they need to exchange a set of pre-defined probe signals (can be any kind) to evaluate the randomness of the channel gain H using Equation 8.3. In order to have a low mismatch rate, they must exchange each probe signal within the Coherence Time (T_c) interval. Meanwhile, in order to keep bits of the generated PSK uncorrelated to each other, the time interval defined as τ_{step} between exchanging each probe signal should be no less than T_c. Notice that, as long as the sender A and receiver B share the same τ_{step}, the process of

exchanging pre-defined signals is naturally synchronized. In this chapter, we assume there exists a pre-defined τ_{step} for both A and B. Otherwise, since knowing τ_{step} will not help the attackers to get the generated PSK, we suggest that one solution would be that the sender A and receiver B may make an agreement on τ_{step} through public communication before the PSK generation process.

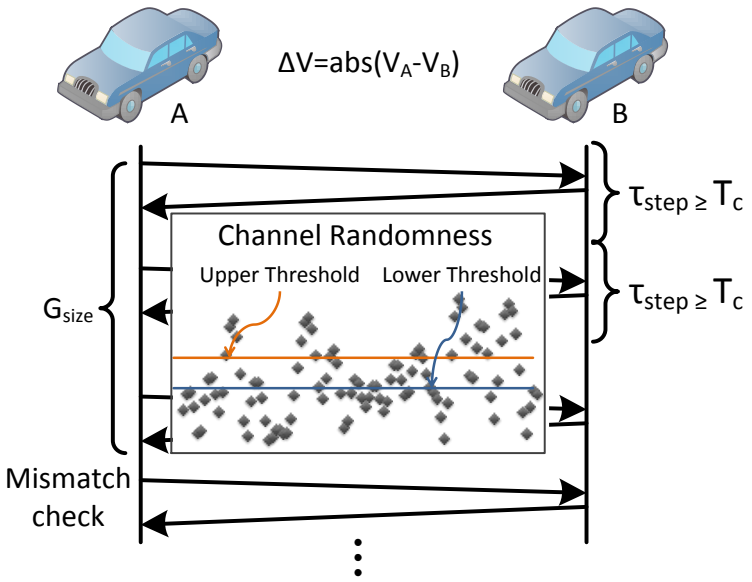


Figure 8.3: Physical Layer PSK Generation for a V2V Scenario

8.3.1 Algorithm Pseudocode and Description

A predefined group of probe signals with a group size G_{size} is sent for evaluating the channel randomness. After the probe signals are exchanged, a set of measured Received Signal Strength (RSS) values is used to generate secret key bits on each side. We then implement a mismatch checking step to remove mismatching bits. During this step, the sender and receiver will publicly exchange the indexes of the probe signals which are used for generating secret bits, in PSK_{idx} and remove the mismatched indexes. Notice that the exchange is public and the attacker may easily get PSK_{idx}. However, the attacker will not be able to figure out the generated bits because only the sender and receiver share the RSSI values of

the probe signals. Once a set of matching bits is generated, the set's size must be greater than or equal to the required PSK length, L_{PSK}. If the set of bits is not long enough, the whole process reiterates until it is. The pseudocode of the wireless channel-based PSK generation algorithm is presented in Algorithm 12.

Algorithm 12: A Wireless Channel-Based PSK Generation Algorithm for Automotive CPS

Input: Measured Signal Strength: RSS
Input: Sample Time Step: τ_{step}
Input: Group Size: L_G
Input: Threshold Parameter: α
Input: Required Pre-Shared Key Length: L_{PSK}
Output: Generated Pre-Shared Key: PSK

```
1  L = 0; PSK = 0; RSS_set=∅; RSS_filtered=∅; PSK_idx=∅;
2  while L < L_PSK do
3      for i=1 to L_G do
4          RSS_set = RSS_set ∪ RSS_i;
5          Wait(τ_step);
6      RSS_filtered = RSS_set * H_highpass(t);
7      MeanValue = Average of RSS_filtered;
8      Var = Variance of RSS_filtered;
9      Th_up = MeanValue + α * Var;
10     Th_lo = MeanValue − α * Var;
11     foreach RSS_j ∈ RSS_filtered do
12         if RSS_j > Th_up then
13             PSK = (PSK << 1) + 0;
14             L = L + 1;
15             Record j in PSK_idx;
16         else if RSS_j < Th_lo then
17             PSK = (PSK << 1) + 1;
18             L = L + 1;
19             Record j in PSK_idx;
20     Exchange PSK_idx;
21     Remove mismatch bits from PSK;
22 return PSK;
```

Lines 3-5 take $(L_G \times \tau_{step})$ time to collect all Received Signal Strength (RSS) values from the wireless channel. Line 6 filters the low frequency parts of the collected RSS values with a high pass filter defined by its impulse frequency response $H_{highpass(t)}$. The filtered signal values

$RSS_{filtered}$ contains all the information that we need to extract the secret bits. Lines 7-10 calculate the thresholds used for generating bits from the received RSS values. As proposed by [136], we use two thresholds. Every RSS value greater than the upper threshold Th_{up} is considered as 1 and every RSS value less than the lower threshold Th_{lo} is considered as 0. Any value in between Th_{up} and Th_{lo} is discarded. The thresholds Th_{up} and Th_{lo} are calculated by the equations in Line 9 and Line 10, respectively, based on the mean and variance values of the collected RSSs. Additionally, we use α, which is a configurable parameter for reducing the bit mismatches due to the existence of noise. If the signal-to-noise ratio of the channel and the transmitters is low, α may need to be set as a higher value to reduce the number of mismatches; otherwise, α may be chosen to be a lower value to improve the performance of the algorithm. Lines 11-19 check all the collected RSSs and generate a PSK, PSK, with length, L. Notice that, Line 15 and Line 19 also record the index of all suitable RSSs for generating secret bits. The indexes in PSK_{idx} from the two communicating automotive wireless nodes are exchanged in Line 20. Then, in Line 21, PSK_{idx} is used to remove all mismatching bits. Finally, if $L >= L_{PSK}$, a PSK is generated among both communicating parties; otherwise, the algorithm will reiterate.

8.4 Pre-Shared Key Length Optimization

Our objectives in creating a scheme to secure wireless communication for the automotive domain includes determining an optimal PSK length according to a scenario. We define a scenario as an instance of communication between an automotive cyber-physical system and another entity (can either be an automotive or non-automotive entity) within a specific setting such as the streets or highway. For example, a scenario can be a vehicle communicating with a tolling device in a highway. In this scenario, the vehicle and tolling device will need to generate a PSK within a small amount of time. [51, 154]. For these reasons, it is necessary to

generate an optimal PSK length for our technique to finish under timing constraints with as much security strength as possible. More specifically, the following characteristics determine a scenario: **1)** Types of communicating parties, **2)** Location, **3)** Fading model (slow or fast) and **4)** Coherence time/length. From these details, we can determine the scenario and key lifetime to generate an optimal PSK length for our proposed physical key generation technique. For the following sections, we provide the scenarios we defined and used as motivating examples for our simulations. These can be altered and extended according to designers.

8.4.1 Scenario Mapping

V2V Scenarios

In V2V, if a scenario is an emergency, a communication session would last less than a second, but if it is not, a session could last to many minutes for traffic efficiency purposes. Therefore, a V2V scenario is based on the severity of the situation as summarized by the following: **1)** Emergency Avoidance (milliseconds), **2)** Emergency Detection (seconds) and **3)** Traffic Efficiency (minutes) [154]. It can be possible to detect the severity of a V2V scenario by the amount of time that has passed since the start of the communication session. To do this, we assume that all initiated V2V communication sessions are first categorized as emergency avoidance to prevent unexpected collisions. After many key refreshes, we can relax the scenario time constraints to generate a longer PSK for higher security strength and longer lifetime.

V2I Scenarios

V2I communication scenarios include communication with roadside units, tolling stations, and traffic lights among others. Infrastructure is generally motionless, implying that to

compute the maximum time of communication, or total latency, we need to take into consideration the overall distance which the vehicle will need to cover before it goes out of the infrastructure's communication range. In general, the highest ideal range for realistic communication is approximately .5 miles (750 meters) to .6 miles (1000 meters). [15, 51, 162]

Computing the optimal PSK length for a scenario therefore requires us to calculate the total latency (see next subsection) and the coherence times corresponding to all possible relative velocities. Although we assume general relative velocities such as approximately 0-45 Miles per Hour (mph) for a street scenario and 0-75 mph for a highway scenario [162], it is important to note that the actual relative velocity is computed before PSK length optimization by the vehicles' embedded devices.

Total Latency

Total latency is the assumed limit to how long two entities in vehicular communication will communicate. For V2V in general, we set the total latency of communication between two vehicles to be 200 milliseconds [154], since we always first assume a collision detection and avoidance scenario. On the other hand, we can subdivide the scenarios of V2I communication based on the setting (streets or highway), where the velocity differences in V2I highway scenarios are vaster than those of V2I street communication. By considering the maximum velocities of each type of scenario, we choose highway and street communication total latencies to be approximately 10 seconds and 40 seconds, respectively.

Fading Models

The decision of the fading model is important in the proposed scheme. We make a reasonable assumption that a modern automotive system includes various sensors, such as speed radar, to provide an estimation of the relative speed between other vehicles. As a result, we propose

Algorithm 13: Scenario Mapping

Input: Scenario: $Scen$
Input: Fading Model: $Model$
Output: Coherence Times: $CoherTimes$
Output: Total Latency: $TotalLat$

```
1  CoherTimes=∅
2  TotalLat = getTotalLatency(Scen)
3  if Model == FastFadingModel then
4      VelocityDiffs = getVelocityDifferences(Scen)
5      foreach VelocityDiff_i ∈ VelocityDiffs do
6          Compute coherence time, CoherTime_i using VelocityDiff_i
7          CoherTimes = CoherTimes ∪ CoherTime_i
8  if Model == SlowFadingModel then
9      Velocities = getVelocities(Scen)
10     foreach Velocities_i ∈ Velocities do
11         Compute coherence length, CoherLength_i using AsphIndex, SpeedOfLight,
           and Bandwidth
12         Compute coherence time, CoherTime_i using Velocity_i and CoherLength_i
13         CoherTimes = CoherTimes ∪ CoherTime_i
14 return CoherTimes, TotalLat
```

a possible solution for selecting the fading model as follows: For velocity differences greater than 5 mph, we use the fast fading model for our scenario mapping technique. From the range of velocity differences, we compute corresponding coherence times using equations 8.4 and 8.5 in Section 8.2.1.

For low velocity differences such as 5 mph or under, we require the slow fading model to compute the coherence times and corresponding optimal PSK lengths. We suggest that for two unmoving communicating parties or an emergency avoidance scenario where the parties have low relative velocity (e.g., approximately 0-5 mph), it would be better to use a stored pre-distributed key to implement the symmetric encryption algorithm instead.

When our algorithm does use the slow fading model, we use equation 8.6 provided in Section 8.2.1 to calculate the coherence times from the coherence lengths and scenario mapping values. To compute the coherence lengths, we use the index of refraction of asphalt (1.635), speed of light, and bandwidth of the channel (5.9 Ghz) [162]. The pseudocode of our scenario

mapping algorithm is given in Algorithm 13.

Moreover, we want to emphasize that the fading model itself is not the contribution of this thesis. They can be dynamically updated using more detailed models to improve the key generation performance. And the decision of the fading model can also be agreed between two communication nodes through public communication before the key generation process.

8.4.2 Algorithm Pseudocode and Description

We can now optimize the PSK length using the scenario mapping function and attack model. For safety-critical and resource-limited devices, we optimize using the timing constraint: $LifeTime$ (the total scenario-based latency or the time until a PSK needs to be regenerated to prevent an attacker from computing it). Additionally, we use a parameter called $Fract$, a dimensionless input which provides the designer an option to adjust the timing constraints based on their unique requirements, such as preventing key generation from taking away valuable communication time (since key generation requires packet exchanges). For the purpose of evaluation, we set it to be .2 but increase it to .4 for 5-10 mph and .7 for 0-5 mph. As an example, with $Fract = .2$, the algorithm will compute an optimal length such that the estimated key generation time will be within 1/5th of the lifetime. Consequently, our algorithm will ascertain out of a set of PSK lengths, the most viable optimal length that satisfies this time constraint ($LifeTime * Fract$). In this case, since a chosen PSK length may not meet the length requirement of the encryption method, we will extend the PSK into valid cryptographic keys through methods discussed in Section 8.5. Our PSK length optimization algorithm is detailed in Algorithm 14 and simulation results for our method are provided in Section 8.6.

In the algorithm, we perform Binary Search to discover the optimal PSK length in terms of generation time under lifetime constraints. We thus require minimum and maximum

Algorithm 14: Algorithm for PSK Length Optimization

Input: Minimum Length: $PSKLenMin$
Input: Maximum Length: $PSKLenMax$
Input: Power to Generate a PSK Bit: $Power$
Input: Attack Strength (128-Bit Keys per Second): $AttackStr$
Input: Scenario: $Scen$
Input: Fraction of the Lifetime: $Fract$
Input: Fading Model: $Model$
Output: PSK Generation Energy Values: $EnergyVals$
Output: Optimal PSK Lengths: $OptPSKLens$
Output: Optimal Lifetimes: $OptLifeTimes$
Output: Optimal PSK Generation Times: $OptPSKGenTimes$

```
Current PSK Length: CurrPSKLen=0
{CoherTimes, TotalLat} = ScenarioMapping(Scen, Model)
foreach CoherTime_i ∈ CoherTimes do
    Perform Binary Search to find optimal PSK length
    while (PSKLenMax >= PSKLenMin) do
        CurrPSKLen = Mid(PSKLenMin, PSKLenMax)
        PSKGenTime = CoherTime_i * CurrPSKLen
        PSKGenEnergy = Power * PSKGenTime
        LifeTime = min(2^(128−CurrPSKLen)/AttackStr, TotalLat)
        if KeyGenTime <= (LifeTime * Fract) then
            OptPSKLen = CurrPSKLen
            OptLifeTime = LifeTime
            OptPSKGenTime = PSKGenTime
            Energy = PSKGenEnergy
            PSKLenMin = CurrPSKLen
        else
            PSKLenMax = CurrPSKLen
    OptPSKLens = OptPSKLens ∪ CurrPSKLen
    OptLifeTimes = OptLifeTimes ∪ LifeTime
    OptPSKGenTimes = OptPSKGenTimes ∪ PSKGenTime
    EnergyVals = EnergyVals ∪ Energy
return OptPSKLens, OptLifeTimes, OptKeyGenTimes, EnergyVals
```

lengths in terms of bits (which we specify as 1 and 128) to set the range of our search space. As the velocity difference (or velocity for slow fading model) increases, we expect that within the time interval of $LifeTime * Fract$, the optimal length also increases. This is because a higher velocity difference (velocity) directly enables a higher PSK bit generation rate. In addition to determining the optimal PSK length, the algorithm also produces the effective and generation times and energy overhead values. Notably, this algorithm can also be customized to compute the optimal PSK length under energy constraints (which can be provided as an extra constraint in Line 16). In general, the energy constraints can be determined based on the power consumption of the vehicle's embedded device when computing a single PSK bit. The optimal PSK length will serve as input for the physical layer PSK generation technique specified in the following section.

8.5 Cryptographic Key Derivation

For cryptographic purposes, we need to generate a cryptographic key with length in accordance to a valid existing encryption scheme. For V2V and V2I communication, we will assume that the encryption scheme is AES-128, as it is a fast and efficient standard symmetric encryption scheme (on the order of 9 microseconds to encrypt and decrypt 60 bytes). [154] For AES-128, the key length must be 128 bits; however, since we are concerned with safety applications in V2X communication, we want to provide our algorithm with an optimized PSK length to minimize overhead. This means that the length of the PSK may be less than 128 bits and therefore requires an additional key derivation step.

To convert the PSK into a longer, cryptographically secure key, we propose the use of either the HMAC Key Derivation Function (HKDF) [97] or the Merkle-Damgard bit padding algorithm [65]. Given the key length and the total latency, one solution to choose between the key derivation functions is provided in Algorithm 15. The HKDF takes in the desired key

length, KeyLen, the pre-shared key, PSK, and some mutual data as the salt, Salt, such as past traffic information (e.g., environmental, lane changes, speed changes, etc.), as input, and it outputs a cryptographic key of KeyLen bits. If there is no prior data exchanged between each other, the two parties can use null as the Salt. The resulting key is created by the HKDF concatenating partial results from a one-way hash function, such as the HMAC-SHA256, on both of the inputs. Putting the inputs into the function together, we can run CryptoKey = HKDF(PSK, Data, KeyLen), where CryptoKey is the resulting 128-bit cryptographic key. On the other hand, it is possible to implement the Merkle-Damgard bit padding algorithm on a PSK with insufficient length to convert it into a 128-bit key. The Merkle-Damgard algorithm pads a 1, then successive 0s, and finally the length of the original PSK to the end such that the length is equal to a desired length. This method is simpler and faster than the HKDF, although it does not necessarily produce strong cryptographic keys.

We want to note that the short key is only used for a short session, where the expiration times of a message are small and key generation algorithms are typically not suitable. Despite the short seed length, the key refresh rate chosen from the key optimization algorithm (see Algorithm 14) will help prevent attackers from easily computing the key while it is being used. By the time the attacker figures out the key and decrypts the message after it has been sent, the communication scenario may have changed considerably (for example, broadcasting of real-time traffic information, and V2V communication for emergency purposes) and the decrypted messages are no longer valid afterwards (since they typically have expiration times associated with them). Further, it is important to note that for scenarios where the timing requirements are not strict or the communication session is relatively long, we can generate keys with longer lengths within acceptable time.

Algorithm 15: Algorithm for Cryptographic Key Derivation

Input: Scenario: $Scen$
Input: PSK: PSK
Input: Salt: $Salt$
Input: Desired Key Length: $KeyLen$
Output: Cryptographic Key: $CryptoKey$

```
if SizeOf(PSK) < 128 and getTotalLatency(Scen) == .2 then
    CryptoKey = Merkle-Damgard(PSK, KeyLen)
else if SizeOf(PSK) < 128 && getTotalLatency(Scen) > .2 then
    CryptoKey = HKDF(PSK, Salt, KeyLen))
else
    CryptoKey = Truncate(128,PSK)
return CryptoKey
```

8.6 Simulation Results

8.6.1 Pre-Shared Key Generation

In this section we first evaluate our wireless channel-based PSK generation technique without optimization, implying that generating a PSK is equivalent to generating the cryptographic key. We used MATLAB [113] to simulate the automotive wireless channel with the following parameters. The average driving speed is set to 37 mph and the coherence length for slow fading is set to 20 meters, or about .01 miles, for an urban environment [183]. The simulation evaluates the key generation time with respect to the relative speed between two communicating nodes (0 to 75 mph in our setup). Moreover, the simulation is conducted with respect to 6 different key sizes (56, 112, 128, 168, 192, 256 bits) proposed by the security standards from NIST [81]. The summarized simulation setup is presented in Table 8.2.

Table 8.2: Setup For Our Key Generation Algorithm

Tested Key Length (bits)	56, 112, 128, 168, 192, 256
Relative Velocity Range (mph)	0 to 75
Average Velocity (mph)	37
Signal to Noise Ratio (dB)	80
Coherence Length (mi)	.01
Group Size (bits)	10

As presented in Figure 8.4, our key generation algorithm has negligible performance (10 to 100 milliseconds) overhead when the relative speed is high due to the fast fading of the wireless channel. This implies that our algorithm can be suitable for various V2X applications and scenarios. On the other hand, for the scenario where the relative speed between two nodes is around zero such as intra vehicle communication, our simulation results show a longer generation time (around 1 to 2 minutes). However, compared to the lifetime of the key, which is typically several hours to even months in these scenarios, several minutes can also be considered as negligible. Although in some cases, several seconds of overhead for generation is not acceptable (e.g., safety related applications), our wireless channel-based key generation algorithm can be optimized using the schemes in Section 8.4.

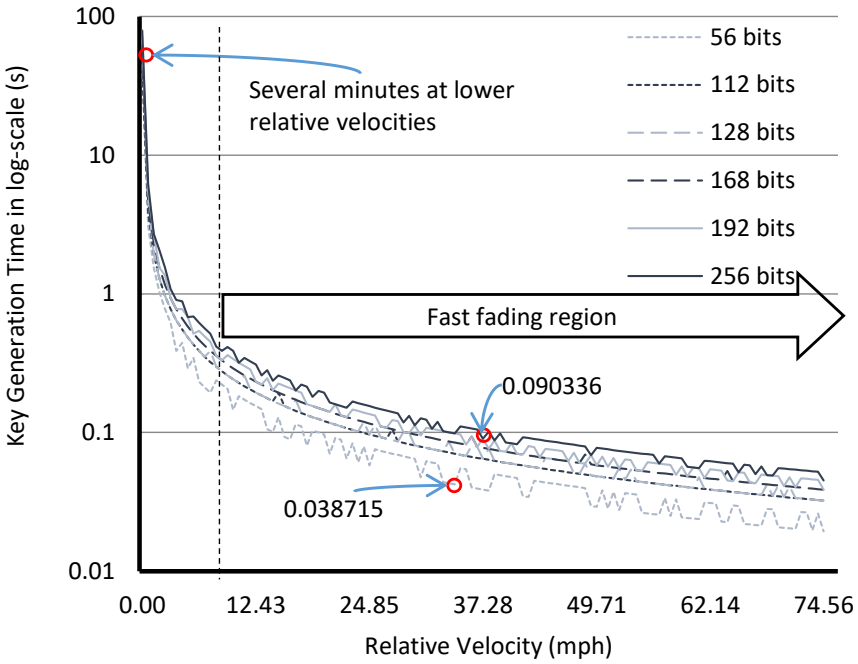


Figure 8.4: Simulation Results of Our Key Generation Overhead

We additionally conducted simulations to confirm the independence of two generated keys from two different automotive wireless communication channels to demonstrate that the attacker cannot easily retrieve the key by eavesdropping. The simulation setup is presented in Figure 8.5. Three vehicles (with driving profiles) are modeled and connected using the

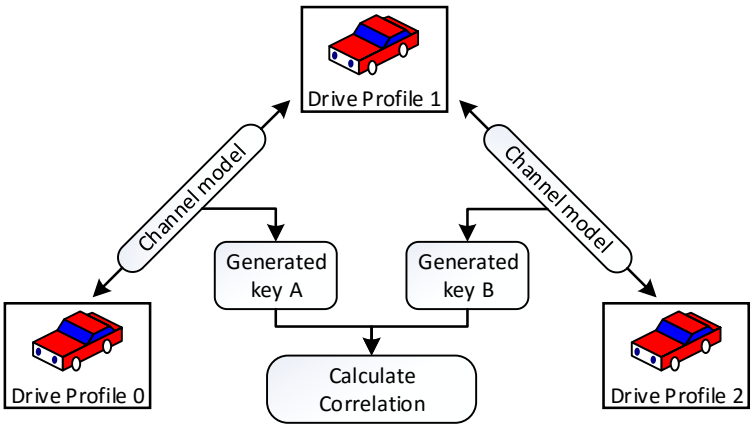


Figure 8.5: Simulation of Generating Two Secret Keys at the Same Time

developed wireless channel models. Two wireless channel models are instantiated in the simulation, where one connects the vehicle models with *Drive Profile 1* and *Drive Profile 0* to each other, and the other connects the vehicle models with *Drive Profile 1* and *Drive Profile 2* with each other.

For each relative speed in mph and key size, as specified in Table 8.3, we run the simulation 100 times to generate two vectors of keys from two wireless channels at the same time. Then, we calculate the Pearson's correlation coefficient [104] between these two vectors. The calculated correlation results are presented in Table 8.3. From the simulation results, we can observe that all the correlation results are close to zero (the highest correlation value is just 0.0392). These results demonstrate the low correlation of keys generated from the channels of two vehicles connected to the same target through wireless communication, thus implying that the attacker cannot retrieve the key generated from the legitimate wireless channel by this method.

Table 8.3: Correlations of the Generated Keys

Key Size / Relative Velocity	56 bits	112 bits	128 bits	168 bits	192 bits	256 bits
0 mph	0.0102	0.0121	0.0132	0.0207	0.0305	0.0233
12 mph	0.0271	0.0053	0.0361	0.0221	0.0337	0.0125
25 mph	0.0264	0.0132	0.0026	0.0125	0.0177	0.0283
37 mph	0.0176	0.0177	0.0056	0.0293	0.0334	0.0268
50 mph	0.0039	0.0236	0.0167	0.0392	0.0147	0.0244

8.6.2 Pre-Shared Key Length Optimization

In this section, we evaluated our PSK length optimization using four example scenarios: V2V/V2I Streets and Highway. From the scenario mapping, we generated the appropriate total latencies, velocity differences or velocities by using the suitable fading models, maximum velocities and incremental values. Furthermore, from these data, we computed the coherence and lifetimes assuming an attacker with low budget and corresponding strength of $2.3 * 10^7$ (128-bit keys per second) [Ecrypt ii]. Finally, we determined the optimal PSK lengths for each scenario to allow PSK generation to occur within a small fraction of the lifetime. Our results are shown in Figures 8.6 and 8.7 and optimization parameters are provided in Table 8.4.

Additionally, as shown in Figures 8.8 and 8.9 we provide the PSK generation time in accordance to optimized PSK lengths. From the figures, it is quite apparent that higher velocity difference implies a lower key generation time. For V2I scenarios, the optimal PSK length of 128 bits can be generated within milliseconds to seconds. As for V2V scenarios, where time is of the essence to detect and prevent collisions, it is apparent that the key generation times must be lower in comparison (up to 100X smaller) to those of the V2I scenarios. Nonetheless, it is possible to create a PSK with 5-50 bits and convert it using our cryptographic key derivation techniques specified in Section 8.5.

Table 8.4: Scenario Mapping Data

Scenario	Total Latency (s)	Velocity Range (mph)	Increment (mph)
V2V Streets	0.2	0-45	3
V2V Highway	0.2	0-75	5
V2I Streets	40	0-45	3
V2V Highway	10	0-75	5

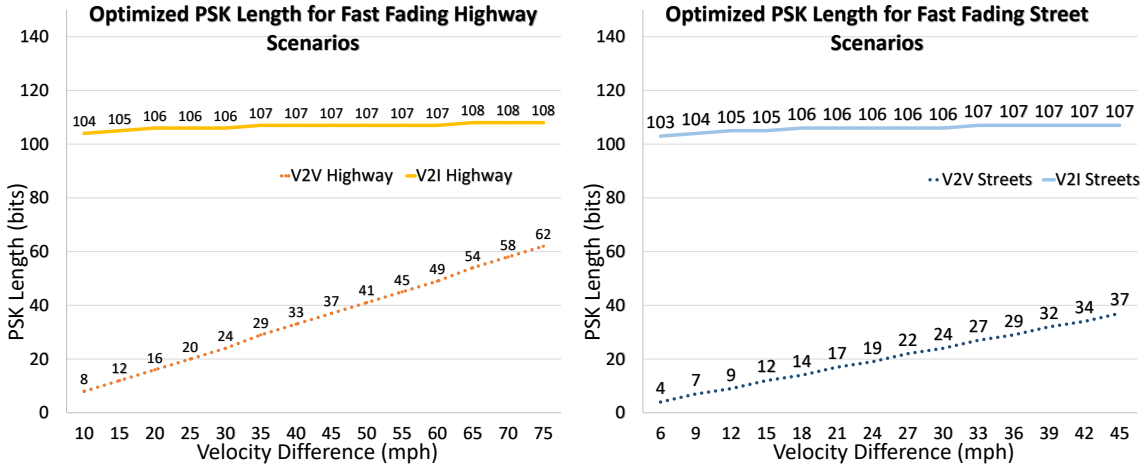


Figure 8.6: Optimized Key Length for Fast Fading Model

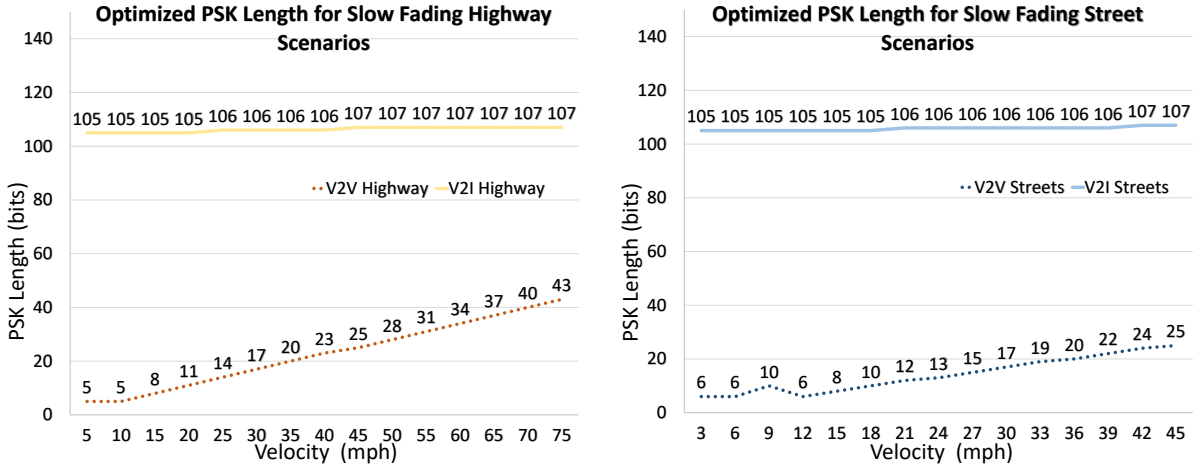


Figure 8.7: Optimized Key Length for Slow Fading Model

8.7 Experimental Results

Going further than simulation, we conducted real world experiments to validate the proposed physical layer key generation technique and **treat the PSK as the cryptographic key to evaluate and demonstrate the practicality of our solution using Algorithm 12.**

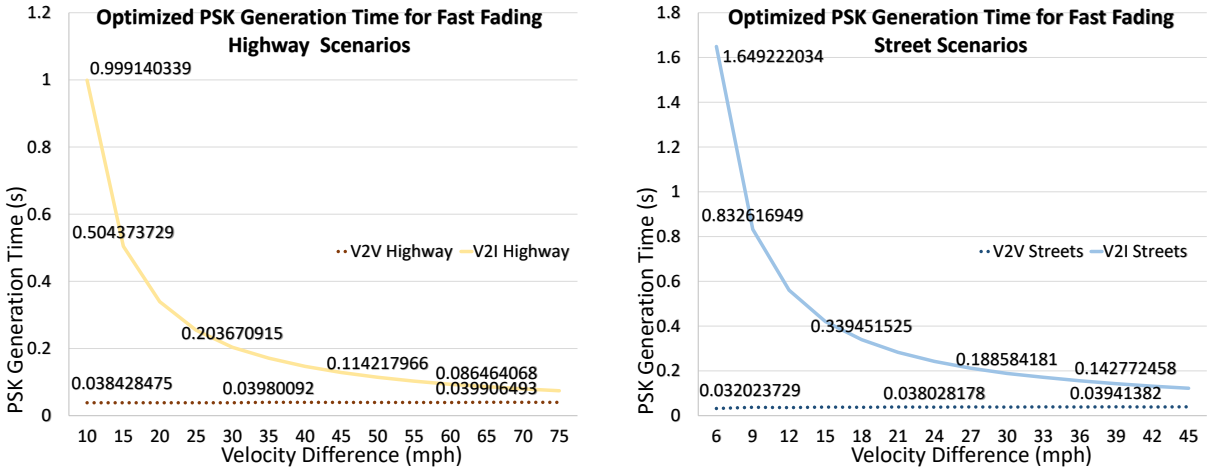


Figure 8.8: Optimized Key Generation Time for Fast Fading Model

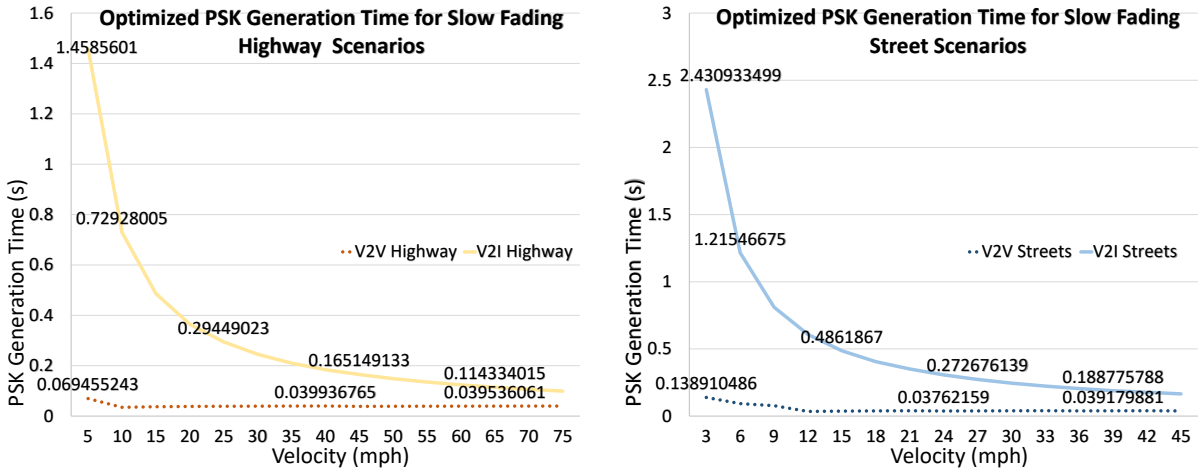


Figure 8.9: Optimized Key Generation Time for Slow Fading Model

8.7.1 Remotely Controlled Car Environment

In our first experiment, we used a system made up of three Remotely-Controlled (RC) cars and Raspberry Pis connected via Bluetooth. As presented in Figure 8.10, we mounted the Raspberry Pi systems on top of the RC cars. On each Raspberry Pi board, we attached Bluetooth dongles (via USB) to establish the wireless communication. One of our objectives was to confirm nearly zero correlation between generated keys from different channels within a short distance, but longer than a few wavelengths (for Bluetooth, the wavelength is approx .125 meters). Therefore, we mounted two Bluetooth dongles on *Car 1* (as shown

in Figure 8.10) to establish two wireless communication channels between *Car 1* and *Car 0*, and *Car 1* and *Car 2*. During runtime, all the Received Signal Strength (RSS) values from each Bluetooth dongle were collected by a computer through separate WiFi channels (as shown in Figure 8.10). For each Bluetooth communication channel, we collected RSS values from the communicating nodes. Thus, in total there were four sets of RSS values collected from all Bluetooth dongles. Although for this experiment we used a computer to execute the key generation algorithm and analyze its results, we can also easily implement the same key generation algorithm in the Raspberry Pis.

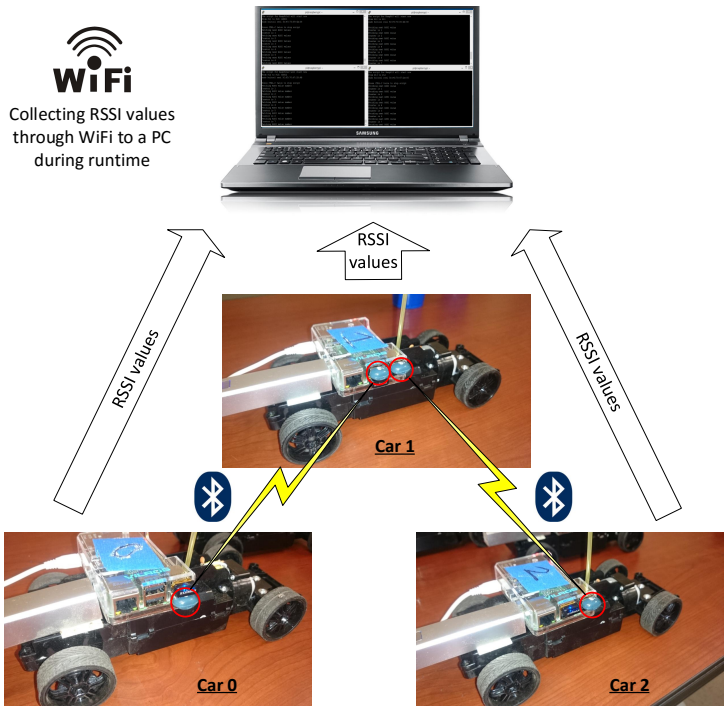


Figure 8.10: RC Car Experiments Setup

We consider the experimental environment with RC cars as a slow fading one because the cars move at low speeds (less than 5 mph) and within a distance of 10 meters (about .006 miles) from each other in open areas with few moving objects around them. 200 samples of the collected RSS values are presented in Figure 8.11. From the results, we can easily observe that the RSS values collected at *Car 1* and *Car 0* for the wireless communication between *Car 1* and *Car 0* are highly correlated with each other (shown in red lines).

The same results are also found for the wireless communication between *Car 1* and *Car 2* (shown in blue lines). These results clearly show the reciprocity characteristic of the wireless communication channel. Moreover, we have found that even with very short distances, the generated RSS values from two different wireless communication channels have nearly zero correlation, thus supporting the assumption that "an attacker that is **several wavelengths** away will experience different wireless channel characteristics, and therefore cannot obtain or predict the keys." Table 8.5 shows the generated 64 bits of keys based on the collected 200 samples of data. Notice that, we use 50 as the probe signal group size for the key generation algorithm in this experiment.

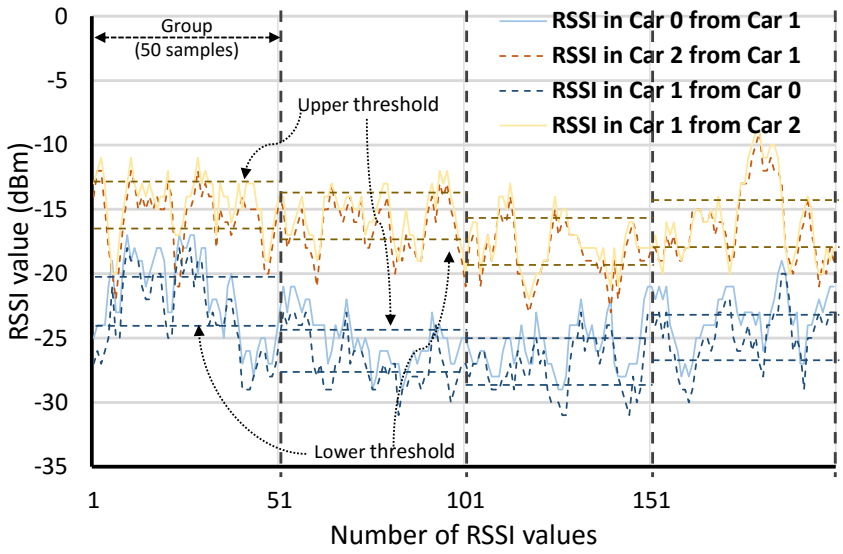


Figure 8.11: Collected Samples of RSS Values from the RC Car Experiments

Table 8.5: Generated 64-bit Keys from the RSS Values

	Generated 64-Bit Keys
Car 1 from Car 0	1100000110000000_0000000100000110_0000000010000000_0000011111111111
Car 0 from Car 1	1100000110000000_0000000100000110_0000000010000000_0000011111111111
Car 1 from Car 2	0000001111111111_1111000000000000_0000011111100000_0000011110000011
Car 2 from Car 1	0000001111111111_1111000000000000_0000011111100000_0000011110000011

8.7.2 Automobile Environment

In order to further validate the practicality of our algorithm, we also performed experiments in real driving scenarios. To do so, we implemented the Bluetooth protocol in our customized applications to acquire RSS values of the wireless channel in real time.

As presented in Figure 8.12, we placed the mobile devices in two vehicles and recorded the RSS values from both vehicles while driving to generate keys. We demonstrate the RSS values received from both sides of the mobile devices during a period in Figure 8.13. We can observe that there exists several mismatched signals in Figure 8.13, this is primarily because Bluetooth communication is not stable between the two fast vehicles resulting in some loss of RSS data. However, our Algorithm 12 already considers these mismatches and handles them well. In this experiment, we would like to note that the RSS value sampling period is 10 milliseconds due to the limitations of the Bluetooth devices (mobile phone and laptop in this experiment). Due to the low sampling rate, each RSS sample will always be obtained after each coherence time, thus prohibiting our algorithm from reaching its full speed.

Figure 8.12: Real World Experiment Using Phones and Laptops

The experiments are conducted based on three relative speeds of 20, 10, and 2 mph with both vehicles moving in the same direction. Throughout the experiment, we collected the RSS values and generated six keys with unique lengths (see Figure 8.14). We want to note

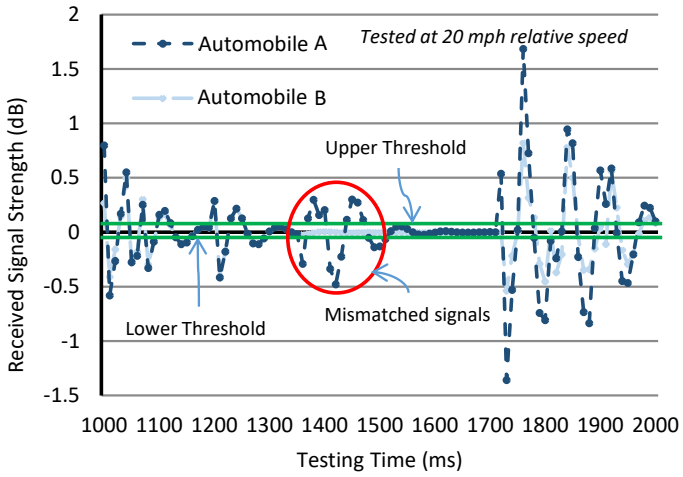


Figure 8.13: Collected RSS Values from Real World Experiment

that sampling takes the majority of time and our key generation algorithm's execution time is negligible (constant).

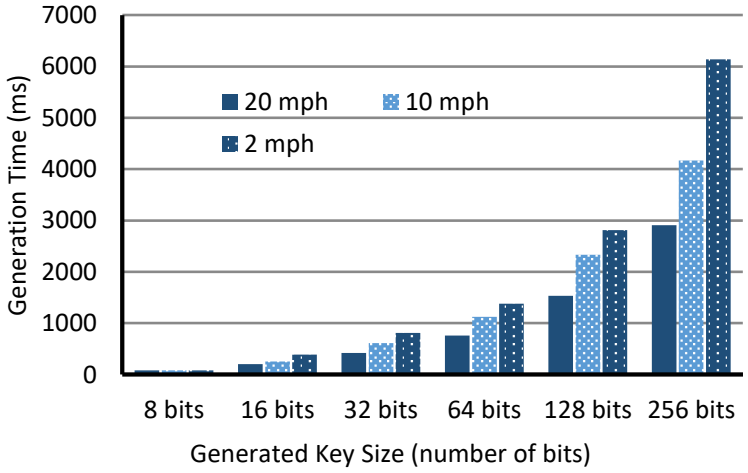


Figure 8.14: Experimental Key Generation Time at Different Speeds

8.8 Evaluation

In this section, we compare our works with the state-of-the-art hybrid cryptographic algorithms [125, 154] to evaluate the security strength, performance and code size overhead for automotive wireless communications.

8.8.1 Security Comparison

We compare the security strength of our algorithm's generated keys to those produced by other techniques. We evaluate and compare the security strength using the proposed average min-entropy as the Key Performance Indicator (KPI). Traditional wireless sensor communication uses pre-distributed keys [129] for their practicality (a simple key management scheme) in achieving real-time communication. However, since the pre-distributed keys and associated algorithms are predictable, the pre-distributed key approaches have little to no entropy [129]. In comparison to the traditional approach, approaches that use PUFs [5], such as the SRAM-PUF [83], can generate keys with high average min-entropy.

To estimate the average min-entropy of our key generation algorithm, we ran our simulation 12800 times to generate $100 * 2^8 = 12800$ number of 8-bit keys. Based on the collected keys, we calculated the probability Pr_{max} of the key with the highest likelihood and applied this Pr_{max} to Equation 8.8. Figure 8.15 shows the resulting average min-entropy of our technique in comparison with other well-known techniques such as pre-distributed keys, Latch-PUFs, DFF-PUFs, and SRAM-PUFs. Note that our algorithm can generate keys with security strength close to that of some of the best PUF-based approaches (up to 67% average min-entropy for 8-bit keys[6]). Although some of the PUF-based approaches (e.g., SRAM-PUF) can generate keys with higher average min-entropy (since the number of 0 and 1 bits tend to be around the same), our algorithm has the advantage of generating keys by directly accessing the communication channel without needing a special physical process such as SRAM rebooting (for SRAM-PUFs). While the average min-entropy (67%) is not as high as some of the PUF-based approaches, it can be potentially increased by adding hardware or algorithm improvements.

[5] A Physical Unclonable Function (PUF) is a function based on physical characteristics that are practically impossible to be duplicated.

[6] According to [176], the average min-entropy increases with the respect to the size of the key.

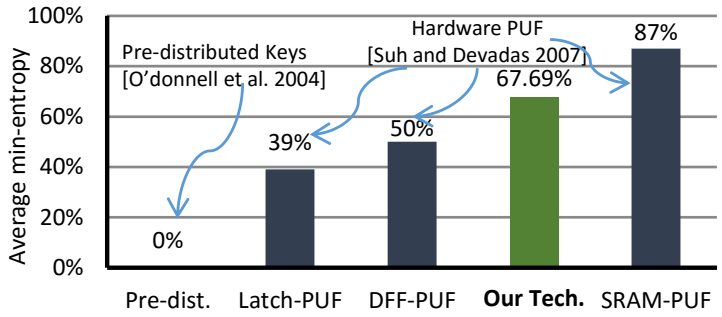


Figure 8.15: Estimated Average Min-Entropy Results Comparison

8.8.2 Performance Overhead Comparison

From the performance point of view, we know that the wireless channel-based key generation algorithm has the advantage of not needing the time-consuming key exchange step of asymmetric and hybrid techniques. Thus, we compare our algorithm's generation time to the execution time of two of the most popular asymmetric cryptographic algorithms (RSA and ECC [73]) used in hybrid solutions [154]. The comparison is conducted given two different NIST security strength (80 and 112 bits) requirements (please refer back to Section 8.2.3 for more details). To evaluate the algorithm's performance, we use the generation time for two different relative velocities (2 mph and 20 mph) collected from our experiments (presented in Section 8.7.2). For the execution times of RSA and ECC algorithms, we have implemented the ECC and RSA algorithms on the Raspberry Pi platform, which has a 1.2GHz 64-bit ARMv8 processor. The results in Table VI shows that at a lower relative speed (2 mph), our algorithm is slower than both RSA and ECC algorithms given the lower security strength (80 bits). However, as the security strength is raised to 112 bits, the performance of the RSA algorithm is significantly reduced and is around 6X slower compared to our algorithm. At 112 bits, the ECC algorithm performs similarly to our algorithm.

On the other hand, at higher relative speed (20 mph), our algorithm has similar performance compared to RSA and ECC algorithms given the security strength of 80 bits. Given the 112 bits security strength, our algorithm becomes 10X faster compared to the RSA algorithm and 1.5X faster compared to the ECC algorithm. The performance of our algorithm for higher

relative speeds mainly depends on how many PSK bits need to be generated and thus has a linear relationship compared to the security strength. Furthermore, given different scenarios in V2X communication where the communication session may last for several seconds or minutes, our Algorithm 3 will be able to find a proper PSK length (which should be small in most scenarios) so that the security requirements will be met while the key generation time is negligible. Notice that in this comparison, we compared our algorithm, which solves both the key generation and exchange problems at the same time, to the key generation and exchange processes in the hybrid algorithm. The current results demonstrate that our proposed technique can be a fair alternative method for V2X communication from a performance perspective.

8.8.3 Code Size Overhead Comparison

In order to evaluate the overhead from the memory size point of view, we also compare the code size of our algorithm to sizes of implemented RSA and ECC algorithms. For a fair comparison, we cross-compiled the code of our proposed key generation algorithm to make it suitable for the same 8-bit processor and to get a valid code size. As shown in Figure 8.6, our algorithm code in comparison is 10X smaller than ECC code and is 20X smaller than RSA code [73]. Additional code, including that of the key length optimization algorithm and the cryptographic key derivation methods, are also negligible in size and easily programmable onto the constrained devices.

Table 8.6: Performance and Code Size Overhead Comparisons Results

Security Strength	Performance Overhead (Seconds)				Code Size Overhead (Bytes)		
	RSA [Raspberry Pi]	ECC [Raspberry Pi]	**Our Alg. (2 mph)**	**Our Alg. (20 mph)**	RSA [Gura]	ECC [Gura]	**Our Alg.**
80 bits	0.35	0.92	1.725	0.95	6292	3682	331
112 bits	13.26	2.06	2.415	1.33	7736	4812	331

8.9 Conclusion

We have presented a physical layer symmetric cryptographic key generation methodology that exploits the randomness of the automotive wireless communication channel. We have also provided an optimization algorithm for the technique in terms of length, time and/or energy according to V2V/V2I scenarios. Our technique is a low-cost solution, in terms of performance and code size, to help solve the challenging key exchange problem and confidentiality requirements in automotive wireless communication. As demonstrated by our results, the proposed algorithm can generate keys with up to 67% average min-entropy. In addition, our proposed technique can achieve up to 10X performance and 20X code size reduction in comparison to the state-of-the-art hybrid cryptographic algorithms. In summary, we developed a simple yet powerful proof of concept for a practical wireless channel-based symmetric cryptographic key generation technique that can coexist with existing state-of-the-art methods to improve the security and performance of automotive CPS.

Bibliography

[1] "Amesim Platform". http://www.lmsintl.com/LMS-Imagine-Lab-Amesim/.

[2] "Beremiz". `http://www.beremiz.org/`.

[3] "SIEMENS SIMATIC PLC". `http://www.automation.siemens.com/`.

[4] "SIMATIC WinAC RTX". `http://www.automation.siemens.com/`.

[5] "Simulink PLC Coder". `http://www.mathworks.com/products/simulink-coder/`.

[6] U. Abelein, H. Lochner, D. Hahn, and S. Straube. Complexity, quality and robustness - the challenges of tomorrow's automotive electronics. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE'2012*, pages 870–871, 2012.

[7] Aberdeen Group. System design: New product development for mechatronics. 2008.

[8] A. Alvarez Cabrera, K. Woestenenk, and T. Tomiyama. "An architecture model to support cooperative design for mechatronic products: A control design case". *Mechatronics*, pages 534–547, 2011.

[9] A. A. Alvarez Cabrera, M. S. Erden, and T. Tomiyama. On the potential of function-behavior-state (FBS) methodology for the integration of modeling tools. In *Proceedings of the 19th CIRP Design Conference Competitive Design*, pages 412–419, 2009.

[10] ANSYS Tool kits for Automotive Solutions. `http://www.ansys.com/Industries/Automotive/`, 2014.

[11] AUTOSAR Automotive Open System Architecture. `http://www.autosar.org/`.

[12] B. Azimi-Sadjadi, A. Kiayias, A. Mercado, and B. Yener. Robust key generation from signal envelopes in wireless networks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 401–410. ACM, 2007.

[13] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management-part 1: General (revised. In *NIST special publication*. Citeseer, 2006.

[14] L. Bassi, C. Secchi, M. Bonfe, and C. Fantuzzi. A SysML-based methodology for manufacturing machinery modeling and design. *Mechatronics, IEEE/ASME Transactions on*, pages 1049–1062, 2011.

[15] P. Belanovic, D. Valerio, A. Paier, T. Zemen, F. Ricciato, and C. Mecklenbrauker. On wireless links for vehicle-to-infrastructure communications. *Vehicular Technology, IEEE Transactions on*, 59(1):269–282, Jan 2010.

[16] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[17] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl. View consistency in architectures for cyber-physical systems. In *Proceedings of the ACM/IEEE 2nd International Conference on Cyber-Physical Systems, ICCPS'2011*, pages 151–160, 2011.

[18] M. Bloch, J. Barros, M. R. Rodrigues, and S. W. McLaughlin. Wireless information-theoretic security. *IEEE Transactions on Information Theory*, pages 2515–2534, 2008.

[19] S. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled rfid device. In *USENIX Security*, volume 5, pages 1–16, 2005.

[20] S. P. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. "Linear matrix inequalities in system and control theory". *SIAM*, 1994.

[21] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 159–164. ACM, 2006.

[22] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[23] M. Broy, S. Chakraborty, S. Ramesh, M. Satpathy, S. Resmerita, and W. Pree. "Cross-layer analysis, testing and verification of automotive control software". In *Proceedings of the International Conference on Embedded Software, (EMSOFT)*, pages 263–272, 2011.

[24] M. Broy, M. Gleirscher, P. Kluge, W. Krezner, S. Merenda, and D. Wild. Automotive architecture framework: Towards a holistic and standardised system architecture description. Technical report, TUM, 2009.

[25] M. Broy, I. H. Krüger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 2007.

[26] P. D. Bruza and T. Van der Weide. *The semantics of data flow diagrams*. Citeseer, 1989.

[27] C. R. Bryant, R. B. Stone, D. A. Mcadams, T. Kurtoglu, and M. I. Campbell. Concept generation from the functional basis of design. In *Proceedings of International Conference on Engineering Design, ICED'2005*, pages 15–18, 2005.

[28] A. A. Cabrera, M. Foeken, O. Tekin, K. Woestenenk, M. Erden, B. D. Schutter, M. van Tooren, R. Babuka, F. van Houten, and T. Tomiyama. "Towards automation of control software: A review of challenges in mechatronic design". *IEEE Transactions on Mechatronics*, pages 876 – 886, 2010.

[29] CAN In Automation and International Users and Manufacturers Group. "CANopen Application Layer and Communication Profile, CiA/DS301, Version 4.2.0". 2000.

[30] A. Canedo, L. Dalloro, , and H. Ludwig. "Pipelining for cyclic control systems". *In Proceedings of the 16th ACM international conference on Hybrid systems computation and control, (HSCC'13)*, pages 223–232, 2013.

[31] A. Canedo and M. A. A. Faruque. "Towards parallel execution of IEC 61131 industrial cyber-physical systems applications". *in IEEE/ACM Design Automation and Test in Europe (DATE'12)*, pages 554–557, 2012.

[32] A. Canedo, M. A. A. Faruque, and J. H. Richter. "Multi-disciplinary integrated design automation tool for automotive cyber-physical systems". In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, page 315, 2014.

[33] A. Canedo, E. Schwarzenbach, and M. A. Al Faruque. "Context-sensitive synthesis of executable functional models of cyber-physical systems". *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 99–108, 2013.

[34] A. Canedo, E. Schwarzenbach, and M. A. A. Faruque. Context-sensitive synthesis of executable functional models of cyber-physical systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, ICCPS'2013*, pages 99–108, 2013.

[35] A. Canedo, J. Wan, and M. A. A. Faruque. Functional modeling compiler for system-level design of automotive cyber-physical systems. In *Proceedings of the the ACM/IEEE International Conference on Computer-Aided Design, ICCAD'2014*, 2014.

[36] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[37] F. E. Cellier. *Continuous System Modeling*. Springer-Verlag, 1991.

[38] R. N. Charette. "This car runs on code". *IEEE Spectrum*, page 3, 2009.

[39] R. N. Charette. This car runs on code. In *IEEE Spectrum*, 2009.

[40] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[41] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. *USENIX Security Symposium*, 2011.

[42] E. Christen and K. Bakalar. VHDL-AMS a hardware description language for analog and mixed-signal applications. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, pages 1263–1272, 1999.

[43] M.-C. Chuang and J.-F. Lee. Team: Trust-extended authentication mechanism for vehicular ad hoc networks. *IEEE Systems Journal*, 8(3):749–758, 2014.

[44] F. R. Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[45] C. V. S. C. Consortium. *Vehicle Safety Communications Project: Task 3 Final Report: Identify Intelligent Vehicle Safety Applications Enabled by DSRC.* National Highway Traffic Safety Administration, Office of Research and Development, Washington, D.C., 2004.

[46] D. J. Cook and L. B. Holder. *Mining graph data*. John Wiley & Sons, 2006.

[47] A. Cooprider. Automotive embedded systems tutorial-part i. Proceedings of the 51th Design Automation Conference, DAC'2014, 2014.

[48] D. A. Cooprider. "Automotive Embedded Systems Tutorial". *Proceedings of the 51st Annual Design Automation Conference (DAC)*, pages 1–23, 2014.

[49] J. B. Dahmus, J. P. Gonzalez-Zugasti, and K. N. Otto. Modular product architecture. *Design Studies*, pages 409–424, 2001.

[50] K. Dar, M. Bakhouya, J. Gaber, M. Wack, and P. Lorenz. Wireless communication technologies for its applications. *Communications Magazine, IEEE*, 48(5):156–162, 2010.

[51] K. Dar, M. Bakhouya, J. Gaber, M. Wack, and P. Lorenz. Wireless communication technologies for its applications [topics in automotive networking]. *Communications Magazine, IEEE*, 48(5):156–162, May 2010.

[52] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[53] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, pages 13–28, 2012.

[54] A. B. Diallo and W. Dhifli. Pgr: A novel graph repository of protein 3d-structures. *Journal of Data Mining in Genomics & Proteomics*, 6(02), 2015.

[55] D. Dumbacher and S. R. Davis. "Building operations efficiencies into NASAs Ares I crew launch vehicle design". *In 54th Joint JANNAF Propulsion Conference*, 2007.

[56] T. ElBatt, C. Saraydar, M. Ames, and T. Talty. Potential for intra-vehicle wireless automotive sensor networks. In *Sarnoff Symposium, 2006 IEEE*, pages 1–4. IEEE, 2006.

[57] M. Erden, H. Komoto, T. V. Beek, V.D'Amelio, E. Echavarria, and T. Tomiyama. A review of funcion modeling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, pages 147–169, 2008.

[58] I. ETSI. Intelligent transport systems (its); security; threat, vulnerability and risk analysis (tvra). Technical report, ETSI TR 102 893, European Telecommunications Standards Institute, 2010.

[59] M. A. A. Faruque and A. M. Canedo. "Intelligent and Collaborative Embedded Computing in Automation Engineering". *in IEEE/ACM Design Automation and Test in Europe (DATE'12)*, pages 344–355, 2012.

[60] P. H. Feiler, D. P. Gluch, and J. J. Hudak. "The architecture analysis & design language (AADL): An introduction". *DTIC Document*, 2006.

[61] G. Fortney. "Model based systems engineering using validated executable specifications as an enabler for cost and risk reduction". *Proceedings of the 2014 Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, 2014.

[62] D. Frede, M. Khodabakhshian, and D. Malmquist. "A state-of-the-art survey on vehicular mechatronics focusing on by-wire systems". *KTH Royal Institute of Technology*, 2010.

[63] U. Gangoiti, M. Marcos, and E. Estevez. "Using cyclic executives for achieving closed loop co-simulation". *Decision and Control and European Control Conference (CDC-ECC '05)*, pages 4785–4790, 2005.

[64] D. Gao, C. Mi, and A. Emadi. "Modeling and Simulation of Electric and Hybrid Vehicles". *Proceedings of the IEEE*, pages 729 – 745, 2007.

[65] S. Goldwasser and M. Bellare. Lecture notes on cryptography, 2001.

[66] J. Gong, D. D. Gajski, and S. Bakshi. "Model refinement for hardware-software codesign". *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, pages 22–41, 1997.

[67] D. Goswami, M. Lukasiewycz, M. Kauer, S. Steinhorst, A. Masrur, S. Chakraborty, and S. Ramesh. "Model-based development and verification of control software for electric vehicles". *Proceedings of the 50th Annual Design Automation Conference (DAC)*, page 96, 2013.

[68] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. "Time-Triggered Implementations of Mixed-Criticality Automotive Software". *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE'12)*, pages 1227–1232, 2012.

[69] D. Goswami, R. Schneider, and S. Chakraborty. "Co-design of cyber-physical systems via controllers with flexible delay constraints". *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 225–230, 2011.

[70] GrabCAD. `https://grabcad.com/`.

[71] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[72] R. K. Gupta and G. De Micheli. "Hardware-software cosynthesis for digital systems". *IEEE Design & Test of Computers*, pages 29–41, 1993.

[73] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Cryptographic hardware and embedded systems-CHES 2004*, pages 119–132. Springer, 2004.

[74] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[75] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[76] J. Harding, G. Powell, R. Yoon, J. Fikentscher, C. Doyle, D. Sade, M. Lukuc, J. Simons, and J. Wang. Vehicle-to-vehicle communications: Readiness of v2v technology for application. Technical report, 2014.

[77] R. Heckmann and C. Ferdinand. "Worst-case execution time prediction by static program analysis". *AbsInt Angewandte Informatik GmbH*, 2004.

[78] B. Helms and K. Shea. Computational synthesis of product architectures based on object-oriented graph grammars. *Journal of Mechanical Design*, page 021008, 2012.

[79] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[80] J. Hirtz, R. B. Stone, D. A. McAdams, S. Szykman, and K. L. Wood. "A functional basis for engineering design: reconciling and evolving previous efforts". *Research in Engineering Design*, pages 65–82, 2002.

[81] J. Hirtz, R. B. Stone, S. Szykman, D. A. McAdams, and K. L. Wood. A functional basis for engineering design: Reconciling and evolving previous efforts. Technical report, NIST, 2002.

[82] D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. In *Advances in Cryptology-CRYPTO 2007*, pages 553–571. Springer, 2007.

[83] D. E. Holcomb, W. P. Burleson, and K. Fu. Power-up sram state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, 2009.

[84] R. M. Ishtiaq Roufa, H. Mustafaa, S. O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, and I. Seskarb. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. *19th USENIX Security Symposium, Washington DC*, pages 11–13, 2010.

[85] S. Jana, S. N. Premnath, M. Clark, S. K. Kasera, N. Patwari, and S. V. Krishnamurthy. On the effectiveness of secret key extraction from wireless signal strength in real environments. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 321–332. ACM, 2009.

[86] J. C. Jensen, D. Chang, and E. A. Lee. "A Model-Based Design Methodology for Cyber-Physical Systems". *Wireless Communications and Mobile Computing Conference (IWCMC'11)*, pages 1666–1671, 2011.

[87] J. Jost and J. Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer, 2008.

[88] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

[89] J.-H. Kim and J.-B. Song. "Control logic for an electric power steering system using assist motor". *Mechatronics*, pages 447–459, 2002.

[90] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[91] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Moderator-Ravi. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*, pages 753–760. ACM, 2004.

[92] H. Komoto and T. Tomiyama. "A framework for computer-aided conceptual design and its application to system architecting of mechatronics products". *Computer-Aided Design*, pages 931–946, 2012.

[93] H. Komoto and T. Tomiyama. A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. *Comput. Aided Des.*, pages 931–946, 2012.

[94] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[95] A. Kossiakoff, W. Sweet, S. Seymour, and S. Biemer. *Systems Engineering Principles and Practice*. Wiley, 2nd edition, 2011.

[96] F. Koushanfar, A.-R. Sadeghi, and H. Seudie. Eda for secure and dependable cybercars: challenges and opportunities. In *Proceedings of the 49th Annual Design Automation Conference*, pages 220–228. ACM, 2012.

[97] H. Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[98] B. Kruse, C. Muenzer, S. Wolkl, A. Canedo, and K. Shea. A model-based functional modeling and library approach for mechatronic systems in SysML. In *ASME 2012 Intl. Design Engineering Technical Conferences, IDETC'2012*, 2012.

[99] T. Kurtoglu and M. I. Campbell. Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping. *Journal of Engineering Design*, 2008.

[100] LabVIEW System Design Software. `http://www.ni.com/labview/`, 2016.

[101] N. Lawson. Highway to hell: Hacking toll systems, 2008.

[102] Lawrence Berkeley National Laboratory - Modelica Buildings Library. `http://simulationresearch.lbl.gov/modelica`.

[103] E. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing, ISORC'2008, 11th IEEE International Symposium on*, pages 363–369, 2008.

[104] J. Lee Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[105] G. Leen and D. Heffernan. "Expanding automotive electronic systems". *Computer*, pages 88–93, 2002.

[106] J. Li, H. Lu, and M. Guizani. Acpn: a novel authentication framework with conditional privacy-preservation and non-repudiation for vanets. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):938–948, 2015.

[107] C.-W. Lin, L. Rao, P. Giusto, J. D'Ambrosio, and A. Sangiovanni-Vincentelli. An efficient wire routing and wire sizing algorithm for weight minimization of automotive systems. *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*, pages 1–6, 2014.

[108] LMS Imagine.Lab AMESim. `http://www.lmsintl.com/`, 2016.

[109] G. Madl and N. Dutt. "Model-based Analysis of Event-driven Distributed Real-time Embedded Systems". *Ph.D. Dissertation, University of California, Irvine*, 2009.

[110] M. Marcos, E. Estevez, N. Iriondo, and D. Orive. "Analysis and validation of IEC 61131-3 applications using a MDE approach". *Emerging Technologies and Factory Automation (ETFA'10)*, pages 1–8, 2010.

[111] S. Mathur, W. Trappe, N. Mandayam, C. Ye, and A. Reznik. Radio-telepathy: extracting a secret key from an unauthenticated wireless channel. *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 128–139, 2008.

[112] Mathworks. Simscape. `http://www.mathworks.com/products/simscape/`.

[113] Matlab. `http://www.mathworks.com/`, 2016.

[114] M. Mazo Jr, A. Davitian, and P. Tabuada. "Pessoa: A tool for embedded controller synthesis". *Computer Aided Verification*, pages 566–569, 2010.

[115] M. C. McFarland, A. C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proceedings of the IEEE*, pages 301–318, 1990.

[116] P. Meerwald and S. Pereira. Attacks, applications and evaluation of known watermarking algorithms with checkmark. In *Proc. SPIE*, volume 4675, page 294, 2001.

[117] S. J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[118] Oregon State University, Design Engineering Lab, Design Repository. `http://designengineeringlab.org/`.

[119] D. Mendez-Fernandez, B. Penzenstadler, M. Kuhrmann, and M. Broy. "A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering". *In Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, pages 183 – 197, 2010.

[120] N. Mhleis, M. Gla, L. Zhang, and J. Teich. "A co-simulation approach for control performance analysis during design space exploration of cyber-physical systems". *SIGBED Rev.*, pages 23–26, 2011.

[121] C. Miller and C. Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.

[122] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle, 2015.

[123] Modelica Association, Modelica Standard Library. `https://modelica.org/libraries/Modelica/`.

[124] Modelon - Vehicle Dynamics Library. `http://www.modelon.com/`.

[125] M. A. Moharrum and A. A. Al-Daraiseh. Toward secure vehicular ad-hoc networks: a survey. *IETE Technical Review*, 29(1):80–89, 2012.

[126] A. P. Moore, R. J. Ellison, and R. C. Linger. Attack modeling for information security and survivability. Technical report, DTIC Document, 2001.

[127] A. Mukherjee, S. A. A. Fakoorian, J. Huang, and A. L. Swindlehurst. Principles of physical layer security in multiuser wireless networks: A survey. *CoRR*, abs/1011.3754, 2010.

[128] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: from dependability to security. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):48–65, 2004.

[129] C. W. O'donnell, G. E. Suh, and S. Devadas. Puf-based random number generation. *In MIT CSAIL CSG Technical Memo*, 2004.

[130] OMG Systems Modeling Language (SysML). `http://http://www.omgsysml.org/`.

[131] G. Pahl, W. Beitz, J. Feldhusen, and K. Grote. *Engineering Design - A Systematic Approach*. Springer, 2007.

[132] F. Paterno. *Model-based design and evaluation of interactive applications*. Springer Science & Business Media, 2012.

[133] N. Patwari, J. Croft, S. Jana, and S. K. Kasera. High-rate uncorrelated bit extraction for shared secret key generation from channel measurements. *IEEE Transactions on Mobile Computing*, 9(1):17–30, 2010.

[134] M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil. "A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems". *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 10, 2013.

[135] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143, 2006.

[136] S. N. Premnath, J. Croft, N. Patwari, and S. K. Kasera. Efficient high-rate secret key extraction in wireless sensor networks using collaboration. *ACM Transactions on Sensor Networks (TOSN)*, page 2, 2014.

[137] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. "Software engineering for automotive systems: A roadmap". *Future of Software Engineering*, pages 55–71, 2007.

[138] Y. Qian and N. Moayeri. Design of secure and application-oriented vanets. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 2794–2799. IEEE, 2008.

[139] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference, DAC'2010*, pages 731–736, 2010.

[140] S. Ramesh and A. Gadkari. "Rigorous model-based design & verification flow for in-vehicle software". *Proceedings of the 48th Annual Design Automation Conference (DAC)*, pages 13–16, 2011.

[141] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):461–491, 2004.

[142] K. Ren, H. Su, and Q. Wang. Secret key generation exploiting channel characteristics in wireless communications. *Wireless Communications, IEEE*, 18(4):6–12, 2011.

[143] K. Richter, M. Jersak, and R. Ernst. "Learning early-stage platform dimensioning from late-stage timing verification". In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 851–857, 2009.

[144] M. Rostami, J. B. Wendt, M. Potkonjak, and F. Koushanfar. Quo vadis, puf?: trends and challenges of emerging physical-disorder based security. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2014 (DATE'14)*, page 352, 2014.

[145] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[146] S. Rudov-Clark and J. Stecki. The language of FMEA: on the effective use and reuse of FMEA data. In *AIAC-13 Thirteenth Australian International Aerospace Congress*, 2009.

[147] M. Rungger, M. Mazo Jr, and P. Tabuada. "Specification-guided controller synthesis for linear systems and safe linear-time temporal logic". *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 333–342, 2013.

[148] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll. "Schedule Integration Framework for Time-Triggered Automotive Architectures". In *Proceedings of the 51st Annual Design Automation Conference (DAC)*, pages 1–6, 2014.

[149] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 458–463. EDA Consortium, 2013.

[150] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[151] A. Sangiovanni-Vicentelli. Defining platform-based design. In *EEDesign of EETimes*, 2002.

[152] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *Eur. J. Control*, pages 217–238, 2012.

[153] A. Sangiovanni-Vincentelli and M. D. Natale. "Embedded System Design for Automotive Applications". *Computer*, pages 42–51, 2007.

[154] T. Schütze. Automotive security: Cryptography for car2x communication. In *Embedded World Conference*. Citeseer, 2011.

[155] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann. Car2x communication: securing the last meter-a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography. In *Vehicular Technology Conference (VTC Fall), 2011 IEEE*, pages 1–5. IEEE, 2011.

[156] M. Senga. "Modeling and analysis of power steering systems". *Massachusetts Institute of Technology*, 1995.

[157] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[158] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava. Non-invasive spoofing attacks for anti-lock braking systems. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 55–72. Springer, 2013.

[159] M. K. Simon and M.-S. Alouini. Digital communication over fading channels. *John Wiley & Sons*, 2005.

[160] Simulink. `http://www.mathworks.com/products/simulink/`, 2016.

[161] R. Sinha, C. J. Paredis, V.-C. Liang, and P. K. Khosla. "Modeling and simulation methods for design of engineering systems". *Journal of Computing and Information Science in Engineering*, pages 84–91, 2001.

[162] L. Stibor, Y. Zang, and H.-J. Reumerman. Evaluation of communication distance of broadcast messages in a vehicular ad-hoc network using ieee 802.11p. In *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 254–257, March 2007.

[163] R. B. Stone and K. L. Wood. Development of a functional basis for design. *Journal of Mechanical Design*, pages 359–370, 2000.

[164] D. Stuart, R. Mattikalli, D. DeLaurentis, and J. Shah. "META II". 2011.

[165] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–12. IEEE, 2013.

[166] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. *Proceedings of the 44th annual Design Automation Conference (DAC'07)*, pages 9–14, 2007.

[167] J. Sztipanovits, T. Bapty, S. Neema, L. Howard, and E. Jackson. Openmeta: a model- and component-based design tool chain for cyber-physical systems. In *Joint European Conferences on Theory and Practice of Software*, pages 235–248. Springer, 2014.

[168] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[169] K. Thramboulidis. "Model-integrated mechatronics-toward a new paradigm in the development of manufacturing systems". *IEEE Transactions on Industrial Informatics*, pages 54–61, 2005.

[170] K. Thramboulidis and G. Frey. "Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation". *Journal of Software Engineering and Applications*, pages 217–226, 2011.

[171] M. Tiller. "Introduction to physical modeling with Modelica". *Springer*, 2001.

[172] E. Tisserant, L. Bessard, and M. de Sousa. "An Open Source IEC 61131-3 Integrated Development Environment". *IEEE Transactions on Industrial Informatics*, pages 183–187, 2007.

[173] S. Uckun. Meta ii: Formal co-verification of correctness of large-scale cyber-physical systems during design. Technical report, Palo Alto Research Center, 2011.

[174] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[175] A. Vachoux, C. Grimm, and K. Einwich. SystemC-AMS requirements, design objectives and rationale. In *Design, Automation and Test in Europe Conference (DATE), 2003*, pages 388–393, 2003.

[176] R. van den Berg. *Entropy analysis of physical unclonable functions*. PhD thesis, MSc. thesis, Eindhoven University of Technology, 2012.

[177] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[178] J. Wan, A. Canedo, A. Faruque, and M. Abdullah. Functional model-based design methodology for automotive cyber-physical systems. *IEEE Systems Journal*, 2014.

[179] J. Wan, A. Canedo, and M. A. A. Faruque. "Functional Model-based Design Methodology for Automotive Cyber-Physical Systems". *IEEE Systems Journal (ISJ)*, 2014.

[180] J. Wan, A. B. Lopez, and M. A. Al Faruque. Exploiting wireless channel randomness to generate keys for automotive cyber-physical system security. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–10. IEEE, 2016.

[181] Q. Wang, H. Su, K. Ren, and K. Kim. Fast and scalable secret key generation exploiting channel phase randomness in wireless networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1422–1430. IEEE, 2011.

[182] A. Wasicek, P. Derler, and E. A. Lee. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[183] C. Weiß. V2x communication in europe–from research projects towards standardization and field testing of vehicle communication technology. *Computer Networks*, 55(14):3103–3119, 2011.

[184] R. Wilhelm, J.Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, and G. B. et al. "The worst-case execution-time problem overview of methods and survey of tools". *ACM Transactions on Embedded Computing Systems (TECS'08)*, pages 1–53, 2008.

[185] K. Woestenenk, H. Tragter, G. M. Bonnema, A. A. A. Cabrera, and T. Tomiyama. "Multi domain design: integration and reuse". *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 519–528, 2010.

[186] S. Wölkl and K. Shea. A computational product model for conceptual design using SysML. *ASME Conference Proceedings*, pages 635–645, 2009.

[187] T. Wongpiromsarn, U. Topcu, and R. M. Murray. "Receding horizon control for temporal logic specifications". *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 101–110, 2010.

[188] P. T. Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.

[189] D. Work, A. Bayen, and Q. Jacobson. Automotive cyber physical systems in the context of human mobility. In *National Workshop on high-confidence automotive cyber-physical systems*, pages 3–4, 2008.

[190] M. Yampolskiy, P. Horvath, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits. Systematic analysis of cyber-attacks on cps-evaluating applicability of dfd-based approach. In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 55–62. IEEE, 2012.

[191] C. Ye, S. Mathur, A. Reznik, Y. Shah, W. Trappe, and N. B. Mandayam. Information-theoretically secret key generation for fading wireless channels. *IEEE Transactions on Information Forensics and Security*, pages 240–254, 2010.

[192] B. Zan, M. Gruteser, and F. Hu. Key agreement algorithms for vehicular communication networks based on reciprocity and diversity theorems. *IEEE Transactions on Vehicular Technology*, 62(8):4020–4027, 2013.

[193] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. "Schedule optimization of time-triggered systems communicating over the FlexRay static segment". *IEEE Transactions on Industrial Informatics*, pages 1–17, 2011.

[194] K. Zeng, D. Wu, A. J. Chan, and P. Mohapatra. Exploiting multiple-antenna diversity for shared secret key generation in wireless networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.