

Verification & Performance Measurement for Transport Protocol Parallel Routing of an AUTOSAR Gateway System

vom Fachbereich Informatik der Technischen Universität Kaiserslautern zur Verleihung des
akademischen Grades Doktor der Ingenieurwissenschaft (Dr.-Ing.)

Genehmigte Dissertation

von

Hassan Mohammad

Dekan:

Prof. Dr. Klaus Schneider

Vorsitzender der Prüfungskommission:

Prof. Dr. Hans Hagen

Berichtersteller:

Prof. Dr.-Ing. habil. Peter Liggesmeyer

Prof. Dr.-Ing. Eric Sax

Datum der wissenschaftlichen Aussprache: 17. Juni 2016

Declaration

Hassan Mohammad

I hereby declare that this thesis, entitled **Verification & Performance Measurement for Transport Protocol Parallel Routing of an AUTOSAR Gateway System**, is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

(Place, Date)

(Signature)

Acknowledgments

This thesis was the result of a research project supported by the company *MBtech Group GmbH & Co. KGaA* in Sindelfingen between 2012 and 2015. The scientific supervision was under the responsibility of the Department of Computer Science and the Fraunhofer Institute for Experimental Software Engineering at the Kaiserslautern University of Technology.

Here, first and foremost, I would like to express my sincere gratitude to my supervisor, *Prof. Dr.-Ing. habil. Peter Liggesmeyer*, head of the research group Software Engineering: Dependability and the Scientific Director of the Fraunhofer Institute for Experimental Software Engineering, for the great guidance, inspiration and supervision he has shown in helping me complete this research. His knowledge, suggestions and experiences have contributed deeply to the results presented in this thesis.

I would like to express my appreciation to my second assessor, *Prof. Dr.-Ing. Eric Sax*, head of the Institute for information processing techniques at the Karlsruhe Institute of Technology, for reviewing this work and providing me with scientific support and professional advises.

My love and gratitude go to my wife, *Zöhre*, whose endless understanding and support made this work possible. I am very grateful also to my parents. Without their constant support, I could never have come to this far.

Abstract

A wide range of methods and techniques have been developed over the years to manage the increasing complexity of automotive Electrical/Electronic systems. Standardization is an example of such complexity managing techniques that aims to minimize the costs, avoid compatibility problems and improve the efficiency of development processes.

A well-known and -practiced standard in automotive industry is AUTOSAR (Automotive Open System Architecture). AUTOSAR is a common standard among OEMs (Original Equipment Manufacturer), suppliers and other involved companies. It was developed originally with the goal of simplifying the overall development and integration process of Electrical/Electronic artifacts from different functional domains, such as hardware, software, and vehicle communication. However, the AUTOSAR standard, in its current status, is not able to manage the problems in some areas of the system development. Validation and optimization process of system configuration handled in this thesis are examples of such areas, in which the AUTOSAR standard offers so far no mature solutions.

Generally, systems developed on the basis of AUTOSAR must be configured in a way that all defined requirements are met. In most cases, the number of configuration parameters and their possible settings in AUTOSAR systems are large, especially if the developed system is complex with modules from various knowledge domains. The verification process here can consume a lot of resources to test all possible combinations of configuration settings, and ideally find the optimal configuration variant, since the number of test cases can be very high. This problem is referred to in literature as the *combinatorial explosion problem*.

Combinatorial testing is an active and promising area of functional testing that offers ideas to solve the *combinatorial explosion problem*. Thereby, the focus is to cover the interaction errors by selecting a sample of system input parameters or configuration settings for test case generation. However, the industrial acceptance of combinatorial testing is still weak because of the deficiency of real industrial examples.

This thesis is tempted to fill this gap between the industry and the academy in the area of combinatorial testing to emphasize the effectiveness of combinatorial testing in verifying complex configurable systems.

The particular intention of the thesis is to provide a new applicable approach to combinatorial testing to fight the *combinatorial explosion problem* emerged during the verification and performance measurement of transport protocol parallel routing of an AUTOSAR gateway. The proposed approach has been validated and evaluated by means of two real industrial examples of AUTOSAR gateways with multiple communication buses and two different degrees of complexity to illustrate its applicability.

List of my Publications

- [1] H. Mohammad and P. Patil. Verification of transport protocol's parallel routing of a vehicle gateway system. *Vehicular 2014, The Third International Conference on Advances in Vehicular Systems, Technologies and Applications, IARIA*, pages 39–45, 2014.
- [2] H. Mohammad and S. M. Shamooun. Handling conflicts to test transport protocol's parallel routing on a vehicle gateway system. *Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE*, pages 1559–1568, 2014.

Contents

List of my Publications	v
List of Figures	1
List of Tables	3
Acronyms	5
1 Introduction	9
1.1 Introduction	9
1.2 Motivation	10
1.3 Contribution	11
1.4 Structure of the Dissertation	11
2 Fundamentals	13
2.1 E/E Systems in Automotive	13
2.2 Vehicle Bus Systems	14
2.2.1 LIN Communication Bus	15
2.2.2 CAN Communication Bus	15
2.2.3 FlexRay Communication Bus	17
2.2.4 MOST Communication Bus	18
2.2.5 Ethernet Communication Bus	19
2.3 Gateway Systems in Automotive	19
2.4 AUTOSAR	21
2.4.1 AUTOSAR Gateway	23
2.4.2 AUTOSAR Transport Protocols	25
3 Transport Protocol Routing of AUTOSAR Gateway	27
3.1 Background	27
3.2 Terminology of TP Routing	29
3.3 Definitions	31
3.4 TP Routing Paradigms of an AUTOSAR Gateway	35
3.4.1 Segmented CAN to CAN TP Routing with Normal Addressing	35
3.4.2 Segmented CAN to FlexRay TP Routing with Normal Addressing	39

3.4.3	Other TP Routing Paradigms	43
3.5	TP Parallel Routing of an AUTOSAR Gateway	46
3.6	The Combinatorial Explosion Problem	49
4	Software Testing	53
4.1	Introduction	53
4.2	Testing Based on Actual Execution	54
4.3	Testing Based on Methodology	54
4.4	Testing Based on Granularity Level	55
4.5	Specification Based Testing	55
4.6	Random Testing	60
4.7	Search Based Testing	60
5	Combinatorial Testing	61
5.1	Introduction	61
5.2	Input Parameter Model (IPM)	62
5.2.1	Interaction Model	64
5.2.2	Interaction Strength	64
5.2.3	Conflict Model	64
5.3	Coverage Level	65
5.4	Conflict Handling Strategy	66
5.5	Combinatorial Approach	67
5.6	Computational Implementation	68
5.7	Test suite Evaluation and Systematic Reduction	69
5.8	Test Case Generation, Test Case Execution and Test Result Evaluation	69
6	Verification & Performance Measurement of Transport Protocol Parallel Routing	71
6.1	Building an IPM	72
6.1.1	Creating an Interaction Model	73
6.1.2	Interaction Strength	77
6.1.3	Creating a Conflict Model	77
6.2	Definition of a Coverage Level	78
6.3	Conflict Handling Strategy	80
6.3.1	Type _A -Conflict Handling	81
6.3.2	Type _B -Conflict Handling	82
6.3.3	Type _C -Conflict Handling	84
6.4	Computational Implementation	85
6.5	Test Suite Reduction	85
6.6	Test Case Generation, Execution and Evaluation	86
6.6.1	Testing TP parallel routing for SNRs	86
6.6.2	Testing TP parallel routing for MNRs	88
7	Validation and Evaluation	91
7.1	Implementation of the Test System	91

7.1.1	Manual Definitions	91
7.1.2	Parameter Abstraction and IPM	92
7.1.3	Test Case selection and Generation Engine	96
7.1.4	Restbus Simulation	97
7.1.5	Analysis	99
7.1.6	Reporting	101
7.2	Gateway Test Object	104
7.3	General Information	104
7.3.1	TP Routing Scenarios	104
7.3.2	Similarity Criteria	105
7.3.3	Conflicts	105
7.4	The First Experiment	106
7.4.1	Results of the Experiment	106
7.5	The Second Experiment	110
7.5.1	Results of the Experiment	111
8	Summary and Outlook	113
8.1	Discussion	113
8.2	Conclusion and Future Work	113
	Bibliography	115

List of Figures

2.1	Automotive E/E Architecture	14
2.2	OSI Reference Model	16
2.3	AUTOSAR Software Architecture Standard	22
2.4	AUTOSAR Gateway Modules	24
3.1	Automotive Distributed Network System	28
3.2	CAN to CAN Simple Frame Routing Example	28
3.3	CAN to FlexRay Frame Routing Example	28
3.4	CAN to CAN segmented Transport Protocol Data Routing	36
3.5	CAN to FlexRay segmented Transport Protocol Data Routing	40
3.6	TP Parallel Routing Example of an AUTOSAR Gateway	47
3.7	E/E System from the TP functionality Point of View	50
5.1	Extended Combinatorial Test Process	63
6.1	V-Model	72
6.2	IPM for TP parallel Routing of AUTOSAR Gateway	73
6.3	Advantages of Building Similar Groups for TP Parallel Routing	76
6.4	Type _A -Conflict Handling Approach	81
6.5	Type _B -Conflict Handling Approach	83
6.6	TP Parallel Routing for SNRs	87
6.7	TP Parallel Routing for MNRs	89
7.1	Test System Design	92
7.2	HTML Report: Description Section	102
7.3	HTML Report: Test Cases Section	103

List of Tables

2.1	Comparison of Communication Buses	20
3.1	CAN Flow Control Frame Content Structure with Normal Addressing	36
3.2	CAN First Frame Content Structure with Normal Addressing	37
3.3	CAN Consecutive Frame Content Structure with Normal Addressing	38
3.4	FlexRay Flow Control Frame Content Structure for Normal Addressing	40
3.5	FlexRay Start Frame Content Structure with Normal Addressing	41
3.6	FlexRay Consecutive Frame Content Structure with Normal Addressing	42
3.7	FlexRay Last Frame Content Structure with Normal Addressing	43
6.1	Example of a <i>Combination Table</i>	79
7.1	First Experiment: First Sub-IPM's Groups	107
7.2	First Experiment: Second Sub-IPM's Groups	108
7.3	First Experiment: Third Sub-IPM's Groups	108
7.4	First Experiment: Forth Sub-IPM's Groups	108
7.5	First Experiment: <i>Combination Table</i> of the first SNR in the first Sub-IPM	109
7.6	First Experiment: An Example of a Resulting Combination	110
7.7	Second Experiment: First Sub-IPM's Groups	111
7.8	Second Experiment: Second Sub-IPM's Groups	111

Acronyms

ABT	Abort
ADC	Analog Digital Converter
ART	Adaptive Random Testing
ARXML	AUTOSAR Extensible Markup Language
AUTOSAR	Automotive Open System Architecture
BC	Bandwidth Control
BfS	Buffer Size
BS	Block Size
BSW	Basic Software
CA	Covering Arrays
CAN	Controller Area Network
CAN FD	CAN Flexible Data Rate
CAN TP	CAN Transport Protocol
CDD	Complex Device Driver
CFT	Consecutive Frame Type
CIT	Combinatorial Interaction Testing
COM	Communication
CP	Category Partition

CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CT	Combinatorial Testing
CTS	Continue To Send
DA	Destination Address
DCM	Diagnostic Communication Manager
DIO	Digital Input/Output
DLC	Data Length Code
DoIP	Diagnostics over Internet Protocol
DTD	Document Type Definition
E/E	Electrical/Electronic
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMI	Electromagnetic Interference
FF_DL	First Frame Data Length
FlexRay TP	FlexRay Transport Protocol
FPL	Frame Payload
FTDMA	Flexible Time Division Multiple Access
I ² C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPDUM	IPDU Multiplexer
IPM	Input Parameter Model
ISO	International Organization for Standardization

LIN	Local Interconnect Network
LIN TP	LIN Transport Protocol
MCAL	Microcontroller Abstraction Layer
ML	Message Length
MOST	Media Oriented Systems Transport
ms	millisecond
N_AE	Network Address Extension
OA	Orthogonal Arrays
OATS	Orthogonal Array Testing System
OBD	Onboard Diagnostic
OS	Operating System
OSI	Open Systems Interconnection
OVFLW	Overflow
PDU	Protocol Data Unit
POF	Plastic Optical Fiber
RT	Random Testing
RTE	Runtime Environment
RTR	Remote Transmit Request
SA	Source Address
SAE	International Standards Organization and Society of Automotive Engineers
Slot_ID	Slot Identifier
SN	Sequence Number
STmin	Minimum Separation Time

SUT	System Under Test
TDMA	Time Division Multiple Access
TP	Transport Protocol
V&V	Validation and Verification
W3C	World Wide Web Consortium
WT	Wait
XML	Extensible Markup Language

Introduction

1.1 Introduction

The competition in automotive market has become very hard in the last decade. Nowadays, customer satisfaction is very essential and can affect directly the automotive industry. The decision to buy a new car is not any more a simple process. Diverse factors may play decisive role in the decision process [1]. Therefore, automotive companies try to win more customers by providing new intelligent and comfort features with feasible costs. Here are some aspects driving very strongly the automotive manufacturer:

- Selling prices.
- Fuel consumption.
- Emissions issues.
- Comfort and convenience features.
- Safety in automotive.
- Time to market.
- After sale services and costs.
- Other innovation features (internet access, mobile communication).
- Competition with other automotive manufacturer.

In order to contribute to the optimization of these aspects, carmakers keep investing in the development of new and innovative technologies. Consequently, the number of features and functionalities continues to increase in modern vehicles. This leads to more complexity of the overall system.

Since innovations in the automotive industry are currently dominated by software based functionalities [2], the software part of the vehicle systems is mostly affected by the complexity issue. It is well-known that the quality of complex software systems is a major problem for the software industry. According to the National Institute of Standards & Technology [3] “the national annual costs of an inadequate infrastructure for software testing is estimated to range from \$22.2 to \$59.5 billion”. Testing is one of the most important means to retain confidence and validate the correctness of software functionalities. However, the decision on the testing technique and the testing thoroughness may affect the overall costs of the development to a wide extend. In order to manage the associated costs of testing and still have an acceptable and measurable coverage, systematic and automatable approaches are used.

This thesis aims to give a scientific research that addresses the software testing problem in highly configurable systems or in systems with a large input space like the automotive central gateway.

1.2 Motivation

As mentioned before, manufacturing costs, time to market and after sale services are leading factors for carmakers. OEMs (Original Equipment Manufacturer) are making noticeable effort to move these factors in a positive direction, either by optimizing processes or by developing new innovative technologies. One of these vehicle’s innovations is the deployment of high end devices and high performance functionalities in E/E (Electrical/Electronic) systems. The present work deals with the testing problem of the high performance functionality of TP (Transport Protocol) parallel routing of AUTOSAR gateways available in modern vehicle’s E/E system. This technology accelerates the production time by allowing parallel flashing of multiple ECUs (Electrical Control Units) in the production phase. It helps also to minimize the maintenance period in after sale services. The optimization gained through TP parallel routing is achieved by reducing the required time to update the vehicle’s software or to inquire the status of various vehicle’s ECUs in parallel, which would then take care for more customer satisfaction.

The implementation of TP parallel routing in automotive field bears a new challenge for testing. This can be seen particularly in complex systems with high configuration grade and large input space, such as the central gateway that represents a central point of communication among the various domains of the vehicle, and to the external world. Such kind of gateways is used currently in most modern vehicle’s E/E systems.

In order to ensure a correct implementation of the TP parallel routing functionality of an AUTOSAR gateway, the test has to cover a large number of possible scenarios. With the increasing number of ECUs that are reachable over the gateway, the number of possible test scenarios is growing exponentially.

Moreover, the testing problem becomes even more complex, since each ECU has also a possible set of configuration parameters that can be modified over the development period.

1.3 Contribution

This thesis is focused on combinatorial testing which is currently an active research topic in the area of software testing. Combinatorial testing consists of various key issues that address the complications related to fighting the combinatorial explosion problem emerged during the testing process. This kind of problems is very common in highly configurable systems or in systems with a big input space.

The contributions of this thesis are focused on the following issues:

- The integration of recent research experiences into a new combinatorial test process.
- Building a creative input parameter model that assists in reducing the number of parameter combinations in the case of testing TP parallel routing of an AUTOSAR gateway system.
- Designing a novel coverage level that helps to avoid similar combinations and increase the test coverage.
- Providing an evidence that combinatorial testing can be effectively applied to complex industrial applications.
- Providing an evidence that combinatorial testing is possible to be automated and is effective in reducing the number of test cases.

1.4 Structure of the Dissertation

The thesis is organized as follows. This chapter gives an overview and scope of the research topic. Chapter 2 includes a technical background on the areas of the handled theme. It introduces the current understandings of E/E architectures in automotive field as well as a briefly description of utilized communication buses. Additionally, automotive gateway systems along with the AUTOSAR standard are explained in chapter 2.

Chapter 3 deals with the transport protocol routing functionality of an AUTOSAR gateway. Transport protocols of the AUTOSAR standard are mainly utilized to meet diagnostic requirements, such as flashing of new software into ECUs of the vehicle or updating their firmware in the after sales services. A deeply investigation of the terms and the supported transport protocol routing paradigms is provided in this chapter to illustrate the functionality. However, the focus is given to the transport protocol parallel routing and the combinatorial explosion problem emerged during the verification and performance measurement process.

In chapter 4, the diverse trends in software testing are stated and discussed. Since the problem dealt with in this thesis is categorized as a combinatorial testing problem, chapter 5 is reserved to discuss the research results on the key areas of combinatorial testing along with the state of the art. Based on this research, a combinatorial test process is consequently proposed to cover the gained knowledge in this area. An implementation methodology is then developed in chapter 6 to apply the proposed combinatorial test process to verify the transport protocol parallel routing of an AUTOSAR gateway and measure the performance. Chapter 7 verifies the

implementation methodology. An evaluation is also given in this chapter to show the results achieved. Finally, chapter 8 completes this work with a summary and outlook.

Fundamentals

In this chapter, the fundamentals of modern vehicle E/E systems are explained briefly. These include the basics of E/E architectures along with their utilized bus systems and involved software. A special attention is given to the vehicle gateways and the AUTOSAR standard, which are nowadays vital for modern vehicle systems.

2.1 E/E Systems in Automotive

An E/E system or E/E architecture is a term used frequently in automotive industry. However, the understandings under this term are wide different from one person to another.

According to the IEEE standard 1471-2000 [4], the architecture of a software-intensive system is defined generally as: “the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”. In automotive industry, an E/E architecture of a vehicle is a system that comprises all software functions, hardware components, the wiring harness and the physical topology. However, some details like the type of a hardware component or the used programming language are not considered as a part of the automotive E/E architecture if they have no impact on other elements.

In the automotive domain, it is common to describe an E/E architecture as a top-down scheme using different views or layers (see fig. 2.1). The first top layer is the Requirements layer which describes all requirements to be met by the E/E architecture. These requirements are realized through a network of functional units in the second layer, the so-called Functional Network layer. The Hardware Component Network layer includes ECUs, bus systems, sensors and actuators realizing functional units from the previous layer. Details about connectors and wiring between the individual components as well as the energy supply are embedded in the Schematic Layer. The layer “Wiring Harness” describes the wiring harness and the lowest layer “Physical Topology” shows in details the physical placement of the ECUs as well as the wiring.

Generally, an automotive E/E architecture can be described as a system that consists of several ECUs operating in different functional domains. For each functional domain, a bus

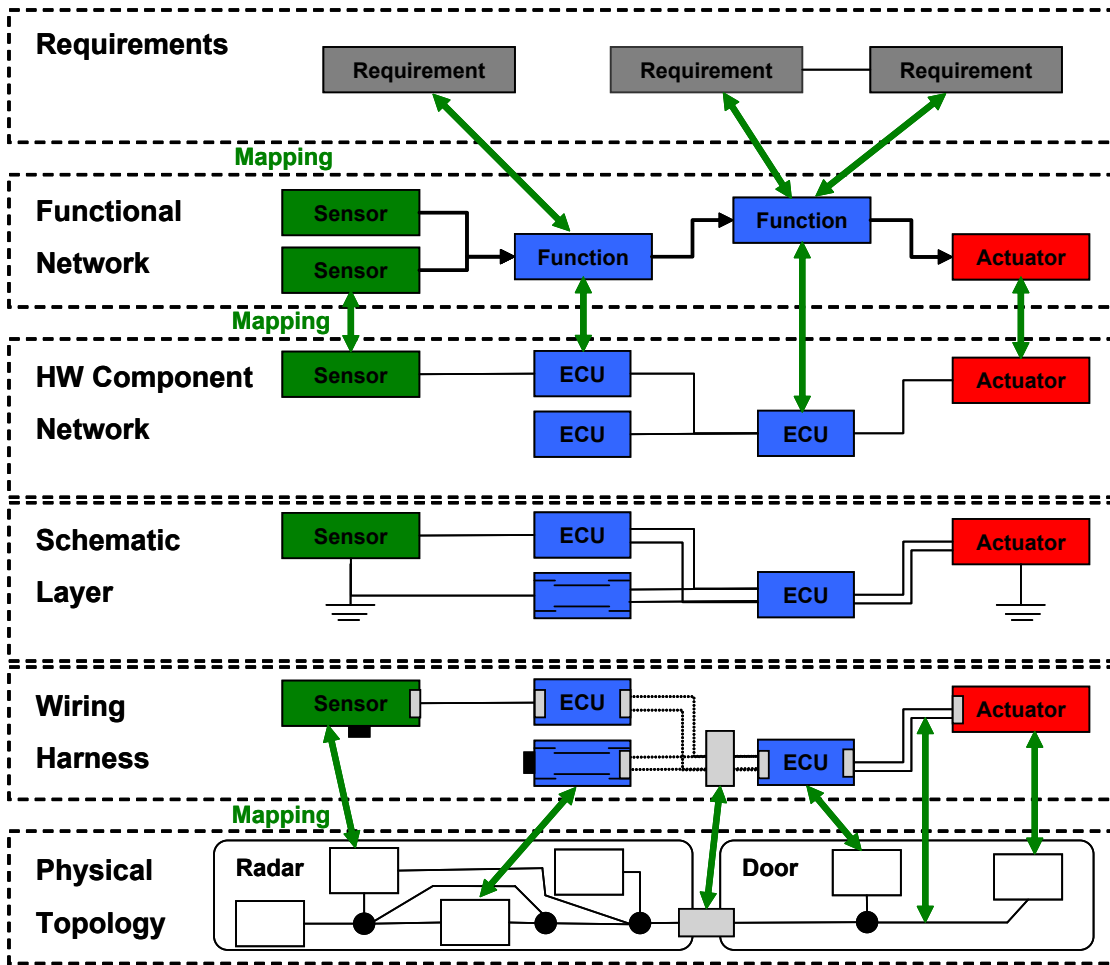


Figure 2.1: Automotive E/E Architecture

system with special characteristics is deployed, and for the data exchange among the functional domains, one or more gateways are responsible [5].

The layered architecture shown in fig. 2.1 is also usual in different design tools that focus on the verification of automotive E/E systems in the development phase, such as PREeVision [6] and Volcano Network Architect [7].

2.2 Vehicle Bus Systems

Today's vehicle E/E systems are designed as distributed systems with a heterogeneous nature in order to manage the increasing complexity and meet the diversity of requirements such as performance, comfort, safety and costs. Therefore, different bus systems are utilized.

In the context of vehicle bus systems, two basic communication paradigms are used. The first paradigm is the time-triggered and the second is the event-triggered.

Within a time-triggered communication, the nodes can access the bus according to a pre-defined time slots, leading to a deterministic behavior during the operation of the system. On the contrary, the bus access in the event-triggered approach can occur at any time during the operation, leading to the advantage of fast reaction to external events which are not known in advance.

In this section, common types of these bus systems are discussed. At the end of the section, a comparison between the explained bus systems is provided (see table 2.1).

2.2.1 LIN Communication Bus

Local Interconnect Network (LIN) was developed as a low cost and low speed (up to 20Kbit/s) serial communication bus for automotive systems. It is mainly used for low performance devices, such as power window, sun roof, rain sensors or seat controllers, where low data transmission rate and a simple fault management are required.

For realizing the communication on the bus, the LIN protocol uses the basics of a time-triggered scheme and a master/slave mechanism. Typically, a system or a subsystem that operates with LIN protocol shall be organized as a cluster that consists of a single master node and one or more slave nodes. The master node manages the message transmissions according to a schedule table and provides the synchronization for all nodes of the cluster.

Scheduling in the LIN protocol is realized through tasks. While the master node contains both a master task and a slave task, each slave node contains only a slave task. The master task uses a polling mechanism to request data from slaves by transmitting a header, which is related to a specific slave task each time it has been sent. Once a header has been transmitted from the master, slave tasks of slave nodes along with the slave task of the master node verify it to determine whether it needs to transmit or receive. If a slave task needs to transmit, it sends one to eight bytes onto the bus followed by a check-sum byte. If the slave task needs to receive, it reads the data and check-sum bytes from the bus.

LIN is standardized through multiple versions. The first standard version completing the byte layer communication was LIN 1.3 [8]. The other versions emerged later (2.0, 2.1, 2.2 and 2.2A) are compatible with LIN 1.3 and supports several additional services and enhancements, such as energy saving through a sleep and wake-up mechanism, simple error detection algorithms, and a bandwidth saving solution.

Currently, LIN is used primarily in low performance body domain application because of its low cost. LIN is planned also to be standardized as ISO standard ISO 17987 Part 1-7.

2.2.2 CAN Communication Bus

Controller Area Network (CAN) bus system was invented originally by Robert Bosch GmbH in 1983, and installed firstly in the late 1980's in Mercedes-Benz cars. The goal was to reduce the wiring harness in automotive E/E systems by developing a serial communication protocol that offers the ability to connect the ECUs directly to one medium.

Reducing the wiring harness has a positive impact on the complexity of the system, since fewer wires have to be modeled and installed. This leads also to the reduction of manufacturing

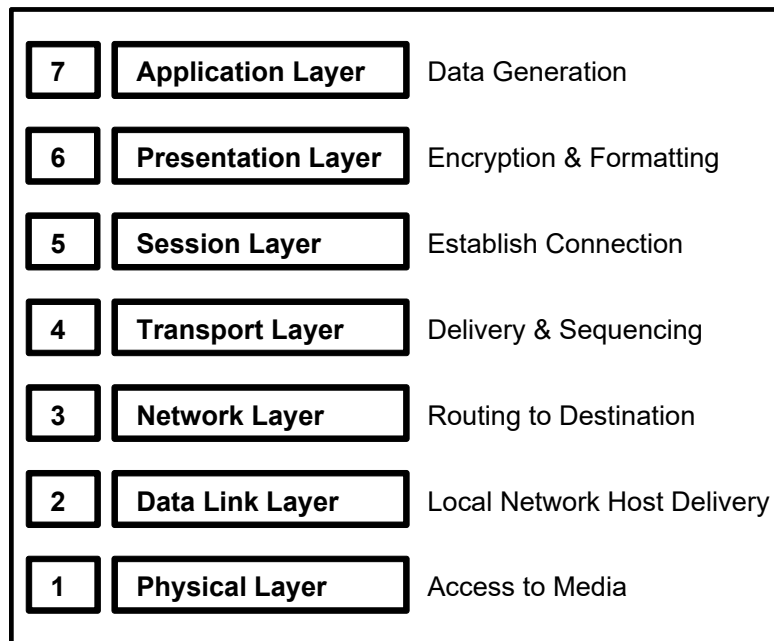


Figure 2.2: OSI Reference Model [9]

costs. Moreover, fewer wires means that the whole weight of the car decreases and this results in a reduction of the fuel consumption.

The CAN communication protocol is asynchronous serial protocol that supports a real time communication in reliable and safe manner. To guarantee real time critical aspects, the release version 2.0B of CAN specification increased the maximum communication rate to 1Mbit/sec. At this rate, even most time-critical messages can be transmitted serially without latency concerns.

In its design, the CAN protocol implements most of the lower two layers of the Open Systems Interconnection (OSI) reference model created by the International Standards Organization (ISO) (see fig. 2.2).

For flexibility purposes, the Bosch CAN specification left out the communication medium portion of the physical layer of the OSI model. As a consequence, some interoperability issues can be raised. To address these issues, the ISO organization in company with the Society of Automotive Engineers (SAE) defined some examples of CAN based protocols, such as ISO11898, ISO11519 and J1939. All these examples include also the definition of the media dependent interface of the physical layer.

Access to the medium in CAN is realized based on the concept of Carrier Sense Multiple Access/ Collision Detection (CSMA/CD) and the technique of non-destructive bit wise arbitration. The identifier field of the CAN message is used for the arbitration process, in which a dominant bit (normally logic bit 0) will always win the arbitration over a recessive bit (normally logic bit 1). Therefore, the lower the value of the identifier field, the higher the priority of it.

In a CAN network, each node must have a unique identifier that is used to identify the node's data and prioritize the node. During the communication, messages are received and eventually

acknowledged by every CAN node of the network. After that, the nodes can decide whether to process the data or discard it. This concept of communication is called message based and is different from address based. In the address based data exchange, messages include the address of the receiving node and only that node will receive and process the message. One benefit of the message based concept is that new nodes can be easily added to the system without the necessity to modify all existing nodes. Another useful feature embedded into the CAN protocol is the Remote Transmit Request (RTR), where nodes can specifically request data to be sent to it.

Generally, four different types of messages or frames are defined in the CAN protocol. These are Data Frames, Remote Frames, Error Frames and Overload Frames. The first two frames are described previously. An Error Frame can be sent by any node that detects one of the errors defined by CAN. However, Overload Frames are generated by nodes that require more time to process messages already received.

For the purpose of data integrity, the CAN nodes have the ability to determine five error conditions and change in one of three error states during the communication. Changing in an error state is based on the detected faults and its count (for more details about error conditions and error states, refers to CAN release version 2.0B [10]).

In 2012, Bosch introduced the CAN FD (CAN Flexible Data Rate) protocol at the international CAN Conference in Germany [11]. In contrast to CAN, the CAN FD provides bit-rates higher than 1 Mbit/s and payloads up to 64 bytes per frame. The main idea of CAN FD is to use different bit-rates in the arbitration and data phase of the communication. That is, once the arbitration has been decided and only one node has to use the bus, the bit-rate can be increased.

It is important to mention that the original CAN FD protocol developed by Bosch has been changed to improve the failure detection capability. The ISO 11898-2:2015 that describes the specification of CAN FD has been already submitted for review.

CAN is used primarily in powertrain, chassis and Body/Comfort application domains of the vehicle. Because of its successful deployment in automobiles, the CAN bus system was established also in other fields, such as medical equipment, avionics (A380), process control and other distributed real-time control systems.

2.2.3 FlexRay Communication Bus

FlexRay was developed jointly by automobile manufacturers and suppliers to deliver deterministic, fault-tolerant and high-speed communication bus required for x-by-wire applications, such as steer-by-wire and break-by-wire. These applications demand more safety, performance and reliability than provided by CAN.

The FlexRay protocol with devices from NXP and Freescale was installed firstly at the end of 2006 in the BMW X5 sport activity vehicle. In 2009, the FlexRay ISO standards ISO 17458-1 to 17458-5 were published as a result of collaboration in the FlexRay consortium. In the mid of 2010, the production reached its maximum. However, the high costs and the emergence of time triggered Ethernet threaten the future use of FlexRay.

FlexRay unifies time- and dynamic event-triggering mechanisms in one protocol, where the communication is based on the Time Division Multiple Access (TDMA) and Flexible Time Division Multiple Access (FTDMA) schemes of networking.

In FlexRay, single and dual channel configurations are supported, where the dual variant offers enhanced fault tolerance and/or increased bandwidth. However, most FlexRay networks utilize only one channel to keep the wiring costs down. The bit rates provided in FlexRay are up to 10 Mbit/s as bus, star and multiple star network technologies. For physical wiring, unshielded twisted pair cabling is used to connect the nodes.

As mentioned previously, time triggered scheme for deterministic data and dynamic event driven scheme for a large variety of frames are supported in FlexRay. The main element of FlexRay is the communication cycle that repeats itself cyclically during system run. After designing the network, the duration of the communication cycle is fixed. The duration value is typically between 1 to 5 ms. Four main parts build up each repeated communication cycle:

- Static segment for deterministic and time triggered data
- Dynamic segment for event triggered data
- Symbol window for maintenance and starting the network
- Network idle time for synchronization purposes

In a FlexRay network, each node is synchronized to the same clock and waits for its turn to write on the bus. The static segment consists of equal-length slots assigned to nodes, where FlexRay nodes may have more than one slot. When a slot occurs in time, the assigned node can transmit its data into that slot. The dynamic segment, which utilizes a scheme similar to the arbitration used by CAN, consists of a defined number of minislots. To prioritize the data during dynamic segment, minislots are assigned to nodes based on message identifiers. A minislot is typically a microsecond long. Higher priority nodes get minislots closer the beginning of the dynamic segment. Once a minislot occurs, the assigned node can transmit its data. If the node misses its minislot, it has to wait until the assigned minislot of the next dynamic segment. While transmitting data during the dynamic segment, future minislots must wait until the node completes its data transmission.

2.2.4 MOST Communication Bus

Media Oriented Systems Transport (MOST) was developed by a consortium of automobile companies and suppliers under the leadership of BMW and DaimlerChrysler. MOST is mainly used for automotive multimedia and infotainment applications, such as video, audio, navigation and telecommunication systems.

MOST supports synchronous and asynchronous data transmission modes along with an additional asynchronous control channel. The synchronous transmission is primarily used for real-time data transmission, like audio/video, with a data bit rates around 25 Mbit/s. For the realization of synchronous transmission, the Time Division Multiplexing (TDM) mechanism is utilized, where the maximum possible number of synchronous data bytes in one MOST frame is 60 bytes. In the case of larger data blocks than 60 bytes, asynchronous transmission with a data bit rate of 15 Mbit/s and a maximum packet length of 1014 bytes can be configured and used.

The additional asynchronous control channel in MOST protocol is mainly used to transfer control data based on the Carrier Sense Multiple Access (CSMA) with a data bit rate of 700 Kbit/s.

The release version 3.0 of MOST specification standard added additional Ethernet and isochronous channels to existing channels of previous specification versions. While isochronous transport mechanism supports extensive video applications with a high bandwidth of 150 Mbit/s, the Ethernet channel is used for efficient transport of IP-based packet data.

MOST networks use the master/slave mechanism to synchronize nodes and establish the communication. One node of a MOST network has to be determined as a master, while all other nodes are slaves. A maximum of 64 nodes arranged in a ring, star or chain topology can be connected to a MOST network. MOST employs Plastic Optical Fiber (POF) as the physical layer for better resilience to Electromagnetic Interference (EMI) and higher data rates.

2.2.5 Ethernet Communication Bus

Ethernet is standardized by the Institute of Electrical and Electronics Engineers (IEEE) 802.3 working group in several standards which define the physical and data link layers. Ethernet together with the higher layer protocol Internet Protocol (IP) are already utilized for some automotive applications such as diagnostics, car-2-car communication, and software flashing. Most of these applications are OEMs specific and not standardized yet. In the area of diagnostics, an International Organization for Standardization (ISO) standard called Diagnostics over Internet Protocol (DoIP) has been released. The standard defines a uniform interface and protocols for diagnostic tester based on Ethernet and IP. Currently, Ethernet is used only for external communication between the vehicle and its environment. There are some studies and researches on using Ethernet as a backbone for intra-communication between the different vehicle domains or for camera links [12]. Ethernet can be a very attractive alternative for existing bus systems in automotive area because of the high throughput provided (10 Mbit/s up to 40 Gbit/s). However, the real time aspects must be studied thoroughly for the usage in some automotive domains. Currently, some solutions for hard real time in automation industry are available [13]. Nevertheless, utilizing them in automobile field should be verified.

2.3 Gateway Systems in Automotive

Over the last years, the increasing number of functionalities in automotive field has led to a high grade of complexity in E/E architectures. Some of the essential factors that contributed to this complexity are the diversity of demands in vehicle functional domains like telematics and safety, as well as the cost aspects. Today's vehicle E/E architectures are distributed in order to overcome this complexity and meet the requirements such as performance, comfort and safety. In a vehicle distributed system, gateways are indispensable. They enable ECUs of connected networks to interchange information necessary for accomplishing specified functionalities. Furthermore, the gateways ensure a comprehensive access through the whole system. Generally, a gateway has to achieve following tasks:

	LIN	CAN	CAN FD	FlexRay	MOST	Ethernet
Functional Domains	Body	Powertrain, Chassis, Body	Powertrain, Chassis, Body	Chassis, x-by-wire	Multimedia, Infotainment	Diagnostics, Car-2-Car, Flashing
Bit Rate	Up to 20 Kbit/s	Up to 1 Mbit/s	Up to 8 Mbit/s	Up to 10Mbit/s	Up to 25 Mbit/s	10 Mbit/s up to 100 Mbit/s
Data Byte Length	Up to 8	Up to 8	Up to 64	Up to 254	Up to 60 for synch., up to 1014 for asynch.	Up to 1500
Physical Layer	Single wire	Twisted pair	Twisted pair	Twisted pair, POF	POF	Twisted pair, Fiber optic
Bus Access	Polling	CSMA/CA	CSMA/CA	TDMA /FDMA	TDM /C-SMA	CSMA/CD, Token ring, Token bus
Topology	Bus	Bus	Bus	Bus, Star, Hybrid	Ring	Bus, Star, Ring, Mesh, Tree

Table 2.1: Comparison of Communication Buses

- **Signal and Frame Routing:** Signals are pieces of information which consist of several bits. They represent data units that cannot be divided. Signal routing is a functionality of the gateway, in which signals are received on one bus and transferred to one or more buses of the connected networks. For the purpose of bandwidth minimization, signals are mostly collected to form one frame. Frame routing is more complex than signal routing. In frame routing, received frames can be routed directly to the destination bus without any manipulation. In other cases, frames may be manipulated in such a way that signals are extracted and new frames are formed and routed to the destination bus.
- **Changing Attributes before Routing:** In some cases, the gateway has to change the attributes of received messages or frames before routing. These attributes are mostly timing aspects. For example, a message is received on one bus sporadic and the gateway sends it cyclic on the destination bus. The signal values of the routed cyclic message can be in this case the last received one or a default one.
- **Transport Protocols:** Transport protocols are utilized in gateways to manage the routing of big data packets which do not fit in one defined frame of the destination network. In this

case, the gateway provides features for segmentation, reassembling, flow control and error detection for routing. Furthermore, the gateway has the ability to convert one transport protocol to another when different types of bus systems and hence transport protocols are involved.

- **Network Management:** For the purposes of energy saving, the gateway manages also the sleep and wake-up behavior of the connected networks. It handles network management requests and responses on all connected networks.
- **Data Filtering:** The gateway has the task of filtering data not required on its connected networks so that the traffic on each network stays at its minimal level.
- **Further Functionalities:** Some gateways has also the ability to collect and save diagnostic information that can be requested later to analyze the behavior and the status of the deployed system.

2.4 AUTOSAR

The Automotive Open System Architecture (AUTOSAR) is an international organization founded in 2003. The aim of the organization is to establish an open standard (the AUTOSAR standard) for the software architecture in vehicle's ECUs. The organization consists of members from OEMs, suppliers, software developers, compiler and semiconductor manufacturers. The goal of the AUTOSAR standard is to improve and facilitate the development of automotive applications by offering the opportunity to reuse soft- and hardware components among different vehicle platforms. To achieve this, AUTOSAR defines a methodology that supports distributed and function-driven development process. The methodology seeks to standardize the software through a layered architecture which consists of the following components as shown in fig. 2.3:

- **Application Software Components:** These are AUTOSAR software components or AUTOSAR Sensor/Actuator components that implement the desired functionalities of the application.
- **Runtime Environment (RTE):** The RTE is a middle ware layer that provides a communication medium between the application software components. Moreover, the RTE enables the application software components to access services of the basic software modules.
- **Basic Software (BSW):** The BSW provides various services to the application software components, such as communication services, mode and memory management services. These services are necessary to run the functional part of the software. The BSW comprises the following modules:
 - **Services:** These are system services, such as diagnostic protocols and memory management services.
 - **Communication (COM):** The COM module provides the communication services over the various bus systems like CAN, LIN, FlexRay and Ethernet. It is also responsible for providing input/output and network management services.

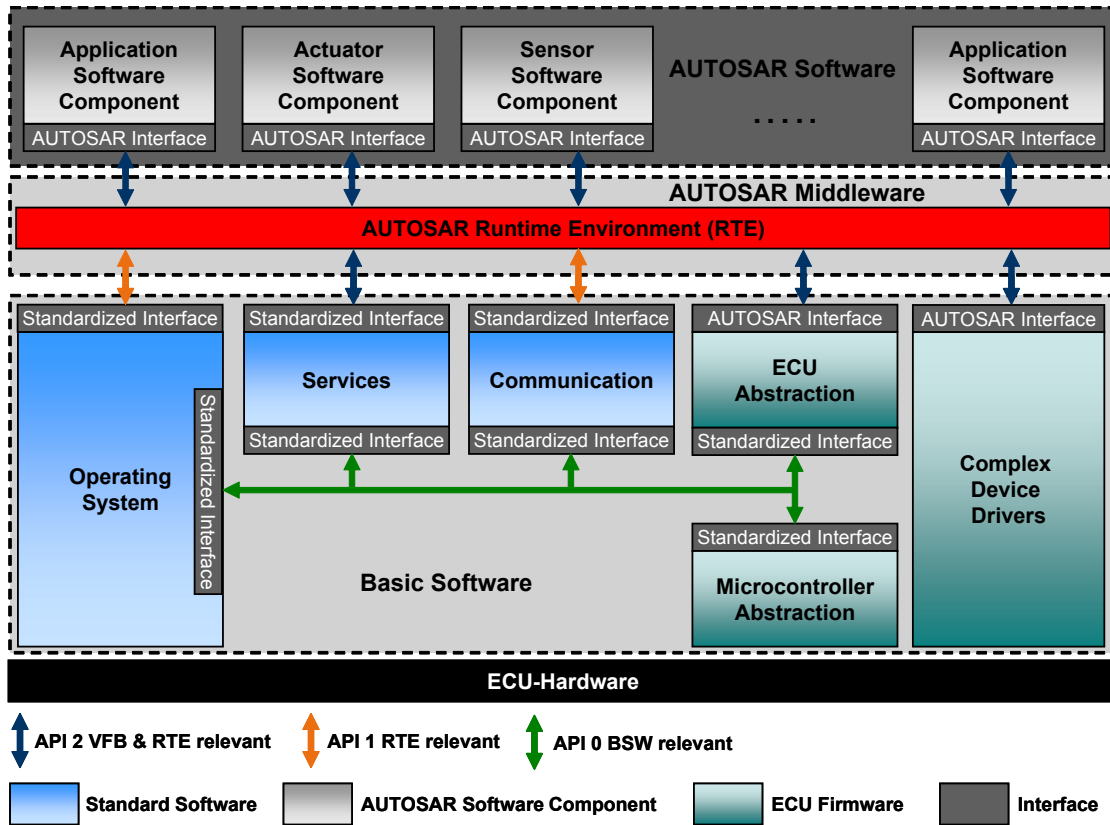


Figure 2.3: AUTOSAR Software Architecture Standard [14]

- **Operating System (OS):** The OS module provides services of task scheduling and resource management.
- **ECU Abstraction:** The ECU abstraction module provides a software interface to the electrical values of any specific ECU in order to decouple higher-level software from all underlying hardware dependencies.
- **Complex Device Driver (CDD):** The CDD is a special module of AUTOSAR that can be configured in various ways to allow a direct access of the higher-layer software components to the ECU hardware or the basic software components. Furthermore, it allows basic software components to directly access the hardware particularly for resource critical applications.
- **Microcontroller Abstraction Layer (MCAL):** The MCAL module is a hardware specific layer that ensures a standard interface to the components of the basic software. It manages the microcontroller peripherals and provides the components of the basic software with microcontroller independent values. MCAL implements notification mechanisms to support the distribution of commands, responses and information to different processes. It routes the access to the hardware to avoid direct

access to microcontroller registers from higher-level software. The MCAL module supports the following services of the microcontroller to the higher layer software components:

- * Digital I/O (DIO)
- * Analog/Digital Converter (ADC)
- * Pulse Width (De)Modulator
- * EEPROM
- * Flash
- * Capture Compare Unit
- * Watchdog Timer
- * Serial Peripheral Interface
- * I²C Bus

2.4.1 AUTOSAR Gateway

An AUTOSAR gateway is a special vehicle ECU that uses the AUTOSAR standard as a basis to implement the common functionalities of a gateway. In an AUTOSAR gateway, two BSW modules are mainly responsible for realizing the routing functionalities. These are the PDU Router and the integral part of the COM module, which is also called Signal Based Gateway. Figure 2.4 shows the position and the interaction relationships of these two modules in the AUTOSAR BSW of a gateway.

PDU Router is the module concerned with PDU routing. During PDU routing, the PDU Router does not change the structure of a PDU. It forwards frames simply as received to the specified destination module. PDU Router is the central BSW module which provides PDU routing between the following modules:

- Communication interface modules
- TP modules
- IPDUM and communication interface modules
- COM module and communication interface modules
- DCM and TP Modules

Basically, the AUTOSAR PDU Router of an AUTOSAR gateway supports two configurable routing paradigms:

- **On-the-Fly Routing:** is a routing mechanism in which frames are routed directly without any delay (the software processing time is not considered as delay).
- **Store-and-Forward Routing:** is a routing mechanism defining that the gateway starts routing after receiving and saving a defined number of frames.

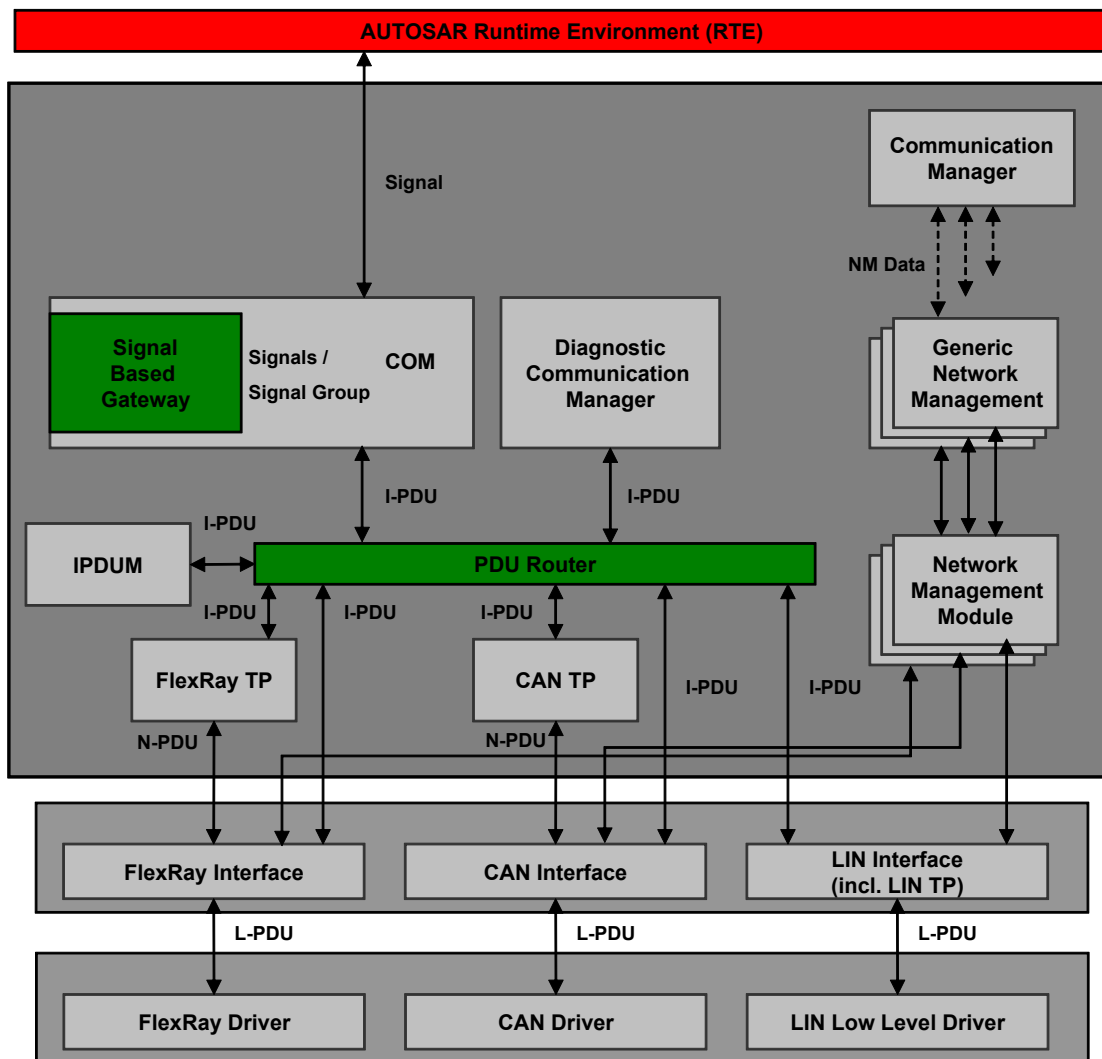


Figure 2.4: AUTOSAR Gateway Modules [14]

PDU Router is a mandatory module instantiated in every AUTOSAR ECU. It provides two different routing schemes. The first routing scheme is based on routing tables and predefined identifiers for configured PDUs. The second scheme provides routing of special PDUs to the DCM module for diagnostic purposes.

The second module involved in the routing process of an AUTOSAR gateway is the Signal Based Gateway, which is part of the COM module. The Signal Based Gateway module is responsible for routing signals and signal groups out of PDUs. To determine the destination of signals and signal groups, the Signal Based Gateway uses unique static names along with a configuration table.

2.4.2 AUTOSAR Transport Protocols

The main functionality of an AUTOSAR gateway is to route data between its connected networks. One type of data routing of the gateway is TP routing. TP routing is required once the data does not fit into one frame of the source or the destination network of the routing process. For TP routing, basic software modules, like CanTP, FrTp and LinTp, are required to realize the routing process. These modules provide the basic features of TP routing, like segmentation, reassembling, flow control and error detection.

In TP routing, the AUTOSAR gateway has also the capability to change the transport protocol of received frames into another transport protocol once the routing involves different types of transport protocols for the source and destination networks, such as routing TP data from a CAN network to a FlexRay network and vice versa.

In the next subsection, the common transport protocol software modules of an AUTOSAR gateway are explained briefly. The availability of these modules depends on the type of networks connected to the gateway. In other words, if the gateway connects a CAN network, a CAN transport protocol module must be available in the software. If the gateway connects also a FlexRay network, a FlexRay transport protocol module must be also available, and so on.

AUTOSAR CAN Transport Protocol

In an AUTOSAR gateway, the basic software module CAN Transport protocol (CAN TP) is responsible for segmenting and reassembling data of the CAN networks that does not fit into the defined CAN frames. The CAN TP module is located between PDU Router and CAN Interface of the AUTOSAR BSW (see fig. 2.4). The functionality of the module is based on the international standard ISO 15765 [15]. This basic software module is required once the AUTOSAR gateway has a connected CAN network. According to AUTOSAR, the CAN TP module has the ability to handle multiple CAN TP connections in parallel. The number of parallel connections depends on the configuration of the module. Configuring more than one connection increases the complexity of the module and has a major effect on resource consumption, as for example processing time and memory usage.

AUTOSAR FlexRay Transport Protocol

As in the case of CAN TP, the basic software module FlexRay Transport Protocol (FlexRay TP) is required to segment and reassemble data that does not fit in the defined FlexRay frame. The FlexRay TP module is located between PDU Router and FlexRay Interface modules of the AUTOSAR BSW (see fig. 2.4). The functionality of the module is based on the international standard ISO 10681-2 [16]. Once the gateway has a connected FlexRay network, this module must be available in the BSW of the AUTOSAR gateway. According to AUTOSAR, the FlexRay TP module has the ability to handle multiple FlexRay TP connections in parallel. The number of parallel connections depends on the configuration of the module. Configuring more than one FlexRay connection increases the complexity of the module and has a major effect on resource consumption, as for example processing time and memory usage.

AUTOSAR LIN Transport Protocol

The functionality of LIN Transport Protocol (LIN TP) is integrated in the AUTOSAR basic software module LIN Interface (see fig. 2.4). Currently, LIN TP parallel routing is not supported in the AUTOSAR LIN TP.

Transport Protocol Routing of AUTOSAR Gateway

3.1 Background

A simple example of a modern vehicle E/E system is depicted in fig. 3.1. As seen in the figure, the system is distributed and hence the functionalities of the system are distributed.

Generally, a modern vehicle E/E system consists of multiple functional domains, such as chassis, powertrain, infotainment, body and diagnostic. Each functional domain is realized as an individual functional network, like the red part of the fig. 3.1 for the body functional domain. A functional network consists of a number of ECUs communicating over a single bus system in order to implement the functionalities of the related domain. Since each functional domain has its own requirements, different communication buses are deployed. As an example, for time critical drive assistant systems, a functional network with FlexRay communication bus is deployed (see the blue part of the fig. 3.1). In other words, a modern E/E system consists of different networks, each communicating over a different bus and hence with a different communication protocol.

For the case that an ECU of a functional network has to exchange data with an ECU located on a different functional network, data routing has to be established. Data routing is the task of a gateway, in which the gateway receives data from one network and sends it to one or multiple networks with or without processing.

The complexity of modern vehicle E/E systems as well as the diversity of communication busses have led to complex and innovative designs that involve multiple gateways such as the central gateway, the telematic gateway and many other more. Nowadays, gateways are indispensable in E/E systems. They enable ECUs within connected networks to interchange information necessary to accomplish specified functionalities. During information interchange, the gateway routes data between its connected networks although they work on different communication protocols.

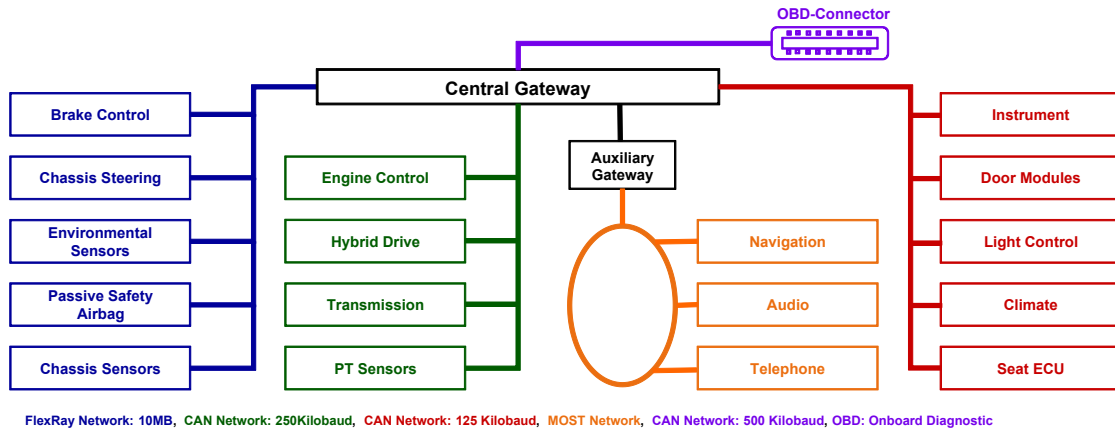


Figure 3.1: Automotive Distributed Network System

Mainly, two types of data routing can be established over the gateway. The first type is frame routing and concerns with routing of data that fits into one frame of the source and destination networks. Figure 3.2 shows an example of a simple frame routing between two CAN networks (Network₁ and Network₂). The gateway in this example receives a cyclic CAN frame on the source CAN network (Network₁) and routes it without any changes (forward) to the destination CAN network (Network₂).

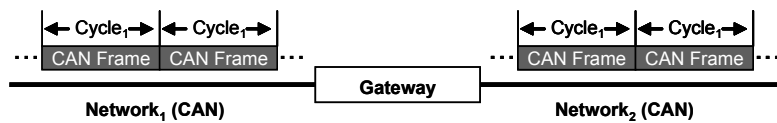


Figure 3.2: CAN to CAN Simple Frame Routing Example

Figure 3.3 represents another paradigm of frame routing, in which the gateway receives multiple cyclic CAN frames holding different information on the source CAN network (Network₁) and routes them together in one FlexRay frame to the destination FlexRay network (Network₂). The routing paradigm in this case is more complex than in the first example, as the gateway

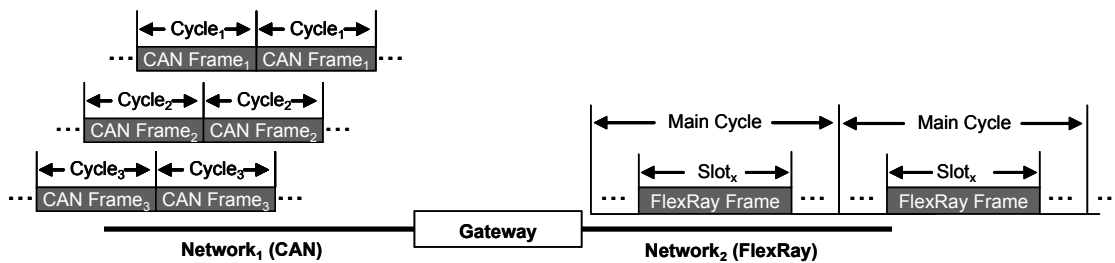


Figure 3.3: CAN to FlexRay Frame Routing Example

has to deal with two different communication protocols (CAN and FlexRay) on the source and destination networks. However, routing in this case remains frame routing, since the data fits in one frame of the source and destination networks.

In more complex examples of frame routing, the gateway can process received frames, i.e., change their attributes or extract data from them and form new frames for routing. Even though, routing remains frame routing.

The second type of data routing is Transport Protocol (TP) routing. It concerns with routing of data packets that do not fit into one frame of the corresponding networks. Routing of large data packets is required for intra-vehicular communication, such as the case of flashing new software over the gateway onto ECUs of the E/E system. For such use case, an external device called External Diagnostic Device is connected via an external interface “OBD-connector” (see fig. 3.1) to the central gateway in order to access and communicate with the ECUs of the connected networks. TP routing is also required for inter-vehicular communication, such as the case of routing large data packets between ECUs of the E/E system.

In TP routing, the gateway utilizes transport protocols. The main purpose of transport protocols is to segment and reassemble data longer than specified message length of the corresponding network. The behavior of the gateway during TP data routing can be different depending on the following aspects:

- Communication protocols of the source and destination networks, e.g., CAN communication protocol and FlexRay communication protocol.
- TP parameters of the source and destination communicating partners. Each ECU implementing TP functionality has a number of TP parameters that specify how the ECU behaves during TP communication.
- Type of routing, i.e., physical routing or functional routing (see section 3.2).
- Size of TP data to be routed.

In the next sections, TP routing of an AUTOSAR gateway is explained in details.

3.2 Terminology of TP Routing

In this section, the terms of TP routing of an AUTOSAR gateway are explained.

- **Routing Channel:** A link of the gateway at which a data routing can take place.
- **TP Routing Channel:** A link of the gateway at which a TP data routing can take place.
- **External Diagnostic Device:** A device which is not permanently connected to the vehicle E/E system. The External Diagnostic Device can be connected to the vehicle for various purposes, e.g. for development, manufacturing and services.
- **OBD-Connector:** A connector to the external environment of the vehicle, to which external devices like External Diagnostic Device can be connected to access the vehicle E/E system.

- **Functional Routing:** Describes the case in which the gateway receives data on one of its connected networks and routes it to multiple other networks. Functional routing can only be unsegmented. This type of routing is also referred to as 'broadcast', 'multicast' or '1 to n routing'.
- **Physical Routing:** Describes the case in which the gateway routes data from a source ECU of one network to a destination ECU of another network. This kind of routing is also referred to as '1 to 1 routing'.
- **Protocol Data Unit (PDU):** A unit of data used for transmission among entities of a vehicle E/E system .
- **Unsegmented routing:** A routing process of the gateway in which the length of data to be routed fits in one frame of the source and destination networks. In this case, no data processing is required by the gateway.
- **Segmented routing:** A routing process of the gateway in which the length of data to be routed does not fit in one frame of the source or destination network. In this case, data processing is required by the gateway.
- **Acknowledged routing:** A routing process of the gateway in which an acknowledgment of frame reception is required from receiving communication partner.
- **Unacknowledged routing:** A routing process of the gateway in which no acknowledgment of frame reception is required from receiving communication partner.
- **Data Length Code (DLC):** A parameter specifies the number of data bytes transmitted in a CAN Frame.
- **Base Identifier:** 11 bits CAN identifier.
- **Extended Identifier:** 29 bits CAN identifier.
- **Request Identifier:** Each ECU implementing TP functionality has a unique Request Identifier used for TP communication.
- **Response Identifier:** Each ECU implementing TP functionality has a unique Response Identifier used for TP communication.
- **Network Address Extension (N_AE):** A parameter used to extend the available address range for large networks, and to encode both sending and receiving network layer entities of subnets other than the local network where the communication takes place.
- **Normal Addressing Format:** An addressing format without Network Address Extension (N_AE) (see [15] [16] for more details).
- **Mixed Addressing Format:** An addressing format with Network Address Extension (N_AE) (see [15] [16] for more details).

- **First Frame:** A CAN TP specific term. It refers to the first frame that the source ECU sends during segmented routing.
- **Flow Control Frame:** A special TP frame used to inform the sending ECU how to behave during the routing.
- **Consecutive Frame:** A special TP frame used to hold the data required to be routed.
- **Block Size (BS):** A CAN TP specific parameter defining the number of consecutive frames (called block) allowed to be send from the source ECU, before waiting for an authorization from the receiver to continue the transmission.
- **Minimum Separation Time (STmin):** A CAN TP specific parameter defining the minimum time the source ECU is to wait between two consecutive frames of a block.
- **Base Cycle:** A FlexRay TP specific parameter defining the offset in cycles for the first occurrence of the respective PDU (see [16] for more details)
- **Cycle Repetition:** A FlexRay TP specific parameter denoting the frequency of a PDU.
- **Slot Identifier (Slot_ID):** An identifier of a specific time slot in the FlexRay schedule.
- **Buffer Size (BfS):** A FlexRay specific parameter similar to Block Size parameter of CAN.
- **N_Bs Timer:** A network layer timing parameter used to control TP communication.
- **N_Cr Timer:** A network layer timing parameter used to control TP communication.

3.3 Definitions

An AUTOSAR gateway is part of a vehicle E/E system (see fig. 3.1). It is a special ECU that can be configured to serve multiple routing channels. Routing channels of the AUTOSAR gateway are used to route data between ECUs of the system, and they are mostly heterogeneous in respect of characteristics and behavior. Generally, a number of ECUs (u) can exchange data over an AUTOSAR gateway in predefined fashions. Each fashion is characterized through a set of configuration parameters. These are required by the gateway to establish routing between communicating ECUs connected to different networks. For TP routing of an AUTOSAR gateway, the following definitions are considered:

- ***TP_Routing_Fashion*** describes a possible routing behavior of TP data and is characterized through a particular set of gateway configuration parameters. An example of a $TP_Routing_Fashion_F$ with P parameters shall be described (3.1) (for possible configuration parameters of CAN TP see [15] and for FlexRay TP see [16]).

$$TP_Routing_Fashion_F = \{P_{F_1}, P_{F_2}, \dots, P_{F_P}\} \quad (3.1)$$

Some common $TP_Routing_Fashions$ with their associated parameters are listed below:

- CAN to CAN TP routing with Normal Addressing

$$\begin{aligned}
 TP_Routing_Fashion_1 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BlockSize, Destination_BlockSize, \\
 & Source_MinimumSeparationTime, \\
 & Destination_MinimumSeparationTime \\
 & \}
 \end{aligned} \tag{3.2}$$

- CAN to CAN TP routing with Mixed Addressing

$$\begin{aligned}
 TP_Routing_Fashion_2 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BlockSize, Destination_BlockSize, \\
 & Source_MinimumSeparationTime, \\
 & Destination_MinimumSeparationTime \\
 & Source_N_AE, Destination_N_AE, \\
 & \}
 \end{aligned} \tag{3.3}$$

- CAN to FlexRay TP routing with Normal Addressing

$$\begin{aligned}
 TP_Routing_Fashion_3 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BlockSize, Destination_BufferSize, \\
 & Source_MinimumSeparationTime, \\
 & Request_Slot_ID, Response_Slot_ID, \\
 & Base_Cycle, Cycle_Repetition, \\
 & \}
 \end{aligned} \tag{3.4}$$

- FlexRay to CAN TP routing with Normal Addressing

$$\begin{aligned}
 TP_Routing_Fashion_4 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BufferSize, Destination_BlockSize, \\
 & Destination_MinimumSeparationTime, \\
 & Request_Slot_ID, Response_Slot_ID, \\
 & Base_Cycle, Cycle_Repetition, \\
 & \}
 \end{aligned} \tag{3.5}$$

- CAN to FlexRay TP routing with Mixed Addressing

$$\begin{aligned}
 TP_Routing_Fashion_5 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BlockSize, Destination_BufferSize, \\
 & Source_MinimumSeparationTime, \\
 & Request_Slot_ID, Response_Slot_ID, \\
 & Base_Cycle, Cycle_Repetition, \\
 & Source_N_AE, Destination_N_AE, \\
 & \}
 \end{aligned} \tag{3.6}$$

- FlexRay to CAN TP routing with Mixed Addressing

$$\begin{aligned}
 TP_Routing_Fashion_6 = \{ \\
 & Request_Identifier, Response_Identifier, \\
 & Source_Network, Destination_Network, \\
 & Source_DLC, Destination_DLC, \\
 & Source_BufferSize, Destination_BlockSize, \\
 & Destination_MinimumSeparationTime, \\
 & Request_Slot_ID, Response_Slot_ID, \\
 & Base_Cycle, Cycle_Repetition, \\
 & Source_N_AE, Destination_N_AE, \\
 & \}
 \end{aligned} \tag{3.7}$$

- CAN to CAN TP functional routing

$$TP_Routing_Fashion_7 = \{ \begin{array}{l} Request_Identifier, \\ Source_Network, Destination_Network, \\ Source_DLC, Destination_DLC, \end{array} \} \quad (3.8)$$

- CAN to FlexRay TP functional routing

$$TP_Routing_Fashion_8 = \{ \begin{array}{l} Request_Identifier, Request_Slot_ID, \\ Source_Network, Destination_Network, \\ Source_DLC, Destination_DLC, \end{array} \} \quad (3.9)$$

- ***TP_Routing_Channel*** is an instance of a *TP_Routing_Fashion*. It has the same set of gateway configuration parameters as in a *TP_Routing_Fashion* and is used by the AUTOSAR gateway to route TP data between particular ECUs. A *TP_Routing_Channel_x* in the *TP_Routing_Fashion_F* shall be described (3.10).

$$TP_Routing_Channel_x = \{P_{F_1x}, P_{F_2x}, \dots, P_{F_Px}\} \quad (3.10)$$

Equations (3.11) and (3.12) are examples of *TP_Routing_Channels* in the *TP_Routing_Fashion₁* “CAN to CAN TP Routing with Normal Addressing”.

$$TP_Routing_Channel_1 = \{0x450, 0x5d9, 1, 2, 8, 8, 32, 8, 0, 10\} \quad (3.11)$$

$$TP_Routing_Channel_2 = \{0x456, 0x5d5, 1, 2, 8, 8, 32, 8, 0, 10\} \quad (3.12)$$

Equation (3.13) is another example of a *TP_Routing_Channel* in the *TP_Routing_Fashion₂* “CAN to CAN TP Routing with Mixed Addressing”.

$$TP_Routing_Channel_2 = \{0x4e9, 0x499, 1, 2, 8, 8, 32, 32, 0, 0, 14, 14\} \quad (3.13)$$

- ***TP_Routing_Scenario*** an AUTOSAR gateway has a number of configured *TP_Routing_Channels* to establish TP routing in different scenarios. Examples of TP scenarios are flashing and Onboard Diagnostic (OBD). *TP_Routing_Scenarios* shall be described as a group of *s* scenarios (3.14).

$$TP_Routing_Scenarios = \{S_1, S_2, \dots, S_s\} \quad (3.14)$$

Equation (3.15) is an example of possible *TP_Routing_Scenarios* of an AUTOSAR gateway.

$$TP_Routing_Scenarios = \{Flashing, Uploading, OBD, Data_Scan\} \quad (3.15)$$

- ***TP_Routing_Instance*** is a relationship between a specific *TP_Routing_Channel* and a possible *TP_Routing_Scenario*. An example of a *TP_Routing_Instance_x* shall be described (3.16).

$$TP_Routing_Instance_x = \{P_{F_{1x}}, P_{F_{2x}}, ..., P_{F_{Px}}, S_x\} \quad (3.16)$$

Equation (3.17) is an example of a *TP_Routing_Instance* (a relationship between *TP_Routing_Channel₁* and the *TP_Routing_Scenario* Flashing).

$$TP_Routing_Instance_1 = \{0x450, 0x5d9, 1, 2, 8, 8, 32, 8, 0, 10, Flashing\} \quad (3.17)$$

3.4 TP Routing Paradigms of an AUTOSAR Gateway

TP routing paradigms are routing models supported by AUTOSAR gateways. Depending on the communication buses of connected networks and the desired behavior, a subset of these paradigms can be configured for a particular AUTOSAR gateway implementation.

In subsections 3.4.1 and 3.4.2, two examples of TP routing paradigms are explained in details. Subsequently, the most well-known and frequently configured TP routing paradigms for AUTOSAR gateways are listed.

The examples explained below do not refer to any error handling mechanisms supported in AUTOSAR TP routing. In addition, the examples are discussed from the gateway point of view.

3.4.1 Segmented CAN to CAN TP Routing with Normal Addressing

Figure 3.4 illustrates the segmented TP data routing of an AUTOSAR gateway between two CAN networks. In this paradigm, a Source ECU located on a CAN network (Source Network) requires to transmit TP data over the Gateway to a Destination ECU located on another CAN network (Destination Network). Due to the same communication protocol of the Source and Destination networks (CAN communication protocol), no protocol change inside the Gateway is required.

During segmented TP data routing, the Gateway (in the middle of fig. 3.4) controls the transmission on the Source and Destination Networks of the routing process. It acts as a receiver on the Source Network and as a sender on the Destination Network. That is, the Gateway receives data from the Source ECU on the Source Network (receiver) and routes it to the Destination ECU on the Destination Network (sender).

Although the Source and Destination ECUs are located on CAN networks, they may have different capabilities regarding processing and memory. Therefore, in order to adjust the capabilities of the Source and Destination ECUs, the Gateway makes use of Flow Control Frames and has to save blocks of CAN messages temporarily in its internal memory by means of defined buffers. Flow Control Frames are used to indicate the current status of the receiver and to control the data routing.

Table 3.1 shows the content structure of a CAN Flow Control Frame with Normal Addressing. Flow Status (FS), which is encoded in the low nibble of Byte 0, can have one of the following values depending on the current status of the receiver:

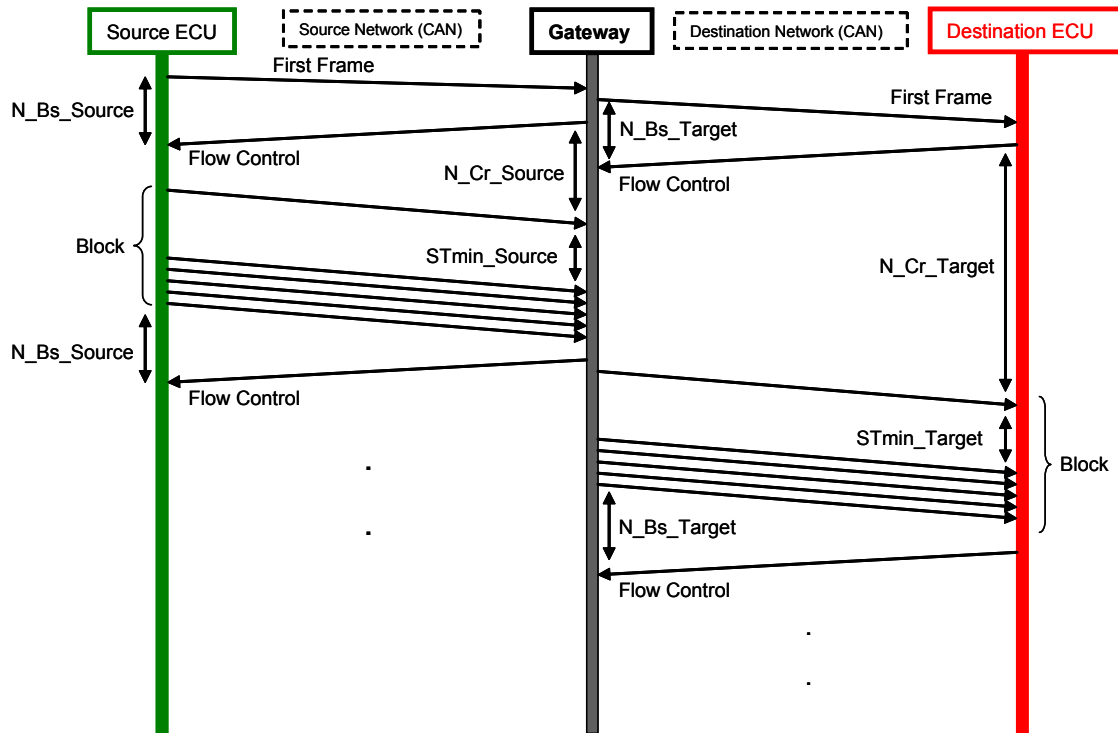


Figure 3.4: CAN to CAN segmented Transport Protocol Data Routing

Byte 0								Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
7	6	5	4	3	2	1	0	BS	STmin	N\A	N\A	N\A	N\A	N\A
0	0	1	1	FS										

Table 3.1: CAN Flow Control Frame Content Structure with Normal Addressing

- **0: Continue To Send (CTS)** indicates that receiving ECU, which can be in this case the Gateway on the Source Network or the Destination ECU on the Destination Network, is able to receive further block of data.
- **1: Wait (WT)** indicates that sending ECU, which can be in this case the Source ECU on the Source Network or the Gateway on the Destination Network, must wait for another Flow Control Frame before proceeding the transmission.
- **2: Overflow (OVFLW)** indicates that receiving ECU, which can be in this case the Gateway on the Source Network or the Destination ECU on the Destination Network, is not able to complete the receiving, and the sender must abort the transmission.
- **3-F: Reserved**

The parameter BS (Block Size), which is encoded in Byte 1 of the CAN Flow Control Frame, defines the maximum number of frames to be sent successively from the sender before waiting

for another Flow Control Frame from the receiver. The parameter STmin (minimum Separation Time), which is encoded in Byte 2, defines a minimum time gap to be hold between two consecutive frames from the sender.

Routing in this paradigm consists of multiple steps on both Source and Destination Networks. These steps are explained below.

On the Source Network

1. The Gateway receives a request frame from the Source ECU on the Source Network as a First Frame. The First Frame must have the content as in table 3.2.

Byte 0								Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
7	6	5	4	3	2	1	0		Data	Data	Data	Data	Data	Data
0	0	0	1	FF_DL										

Table 3.2: CAN First Frame Content Structure with Normal Addressing

The parameter FF_DL (First Frame Data Length) of the First Frame is used to determine the length of data required to be routed in the current session of TP routing. FF_DL can have a value up to FFF (4095) bytes for CAN TP communication.

After receiving the First Frame correctly, one of three states can be raised in the gateway:

- a) The gateway is able to process the First Frame. In this case, following actions are taken:
 - Received First Frame is routed directly without any delay to the Destination ECU on the Destination Network (internal processing time is not considered as a delay).
 - A Flow Control Frame with the value (0) for FS is sent to the Source ECU before a timer N_Bs_Source expires.
 - The Gateway starts a new timer N_Cr_Source on the Source Network to receive a block of data from the Source ECU.
 - The routing proceeds as in step 2.
 - b) The gateway is not able to process the First Frame because no available resources can be reserved for the routing. In this case, a Flow Control Frame with the value (2) for FS is sent to the Source ECU to indicate that routing has been aborted and the Source ECU can send later another routing request. That is, the TP routing is aborted and no further steps are necessary.
2. The gateway receives a block of CAN Consecutive Frames on the Source Network before the timer N_Cr_Source expires. The number of frames of the block and the time gap between them must be as specified in the BS and STmin parameters of received Flow Control Frame on the Source Network. Consecutive Frames must have the content structure as specified in table 3.3

Byte 0								Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
7	6	5	4	3	2	1	0	Data	Data	Data	Data	Data	Data	Data
0	0	1	0	SN										

Table 3.3: CAN Consecutive Frame Content Structure with Normal Addressing

The parameter SN (Sequence Number) in table 3.3 is used to indicate the order of corresponding Consecutive Frame. The parameter SN can take a value between 0 and F and must be incremented by 1 for each further Consecutive Frame. Once the value F has been reached, a wraparound to 0 for the next Consecutive Frame must be occurred.

After a correct reception of a block of CAN Consecutive Frames, routing proceeds as in step 3.

3. The gateway sends a Flow Control Frame to the Source ECU on the Source Network and starts a new timer N_Bs_Source. The FS value of the Flow Control Frame depends on the current status of the gateway and can be one of the following:
 - 0: if the Gateway is able to receive further block. In this case, the routing process proceeds as in step 4.
 - 1: if the reserved buffer is full and the Source ECU must wait for a further Flow Control Frame from the gateway. This is the case when blocks received from the Source ECU have not been yet routed to the Destination ECU, as for example due to slow processing or high bus load on the Destination Network. In this case, the step 3 is repeated for a defined maximum number of attempts. If the maximum allowed number of attempts has been reached, the routing process is aborted and no further steps will be necessary.
 - 2: if the gateway has currently no free memory for further blocks and some timing parameters cannot be hold any more. In this case, the routing process is aborted and no further steps are necessary.
4. The gateway receives further block of CAN Consecutive Frames as specified in the last sent Flow Control Frame before the timer N_Cr_Source expires.
5. The transmission proceeds as in steps 3-4 until the whole data has been routed.

On the Destination Network

If the Gateway was able to process the First Frame on the Source Network, following steps will be taken in the Gateway:

1. The Gateway routes the First Frame directly to the Destination Network.
2. The Gateway starts a new timer N_Bs_Target on the Destination Network to control the reception of a Flow Control Frame from the Destination ECU.

3. The Gateway receives a Flow Control Frame from the Destination ECU with the content structure as specified in table 3.1 before the timer N_Bs_Target expires. The value of parameter FS in the received Flow Control Frame depends on the current status of the Destination ECU and can be one of the following:
 - 0: if the Destination ECU is able to receive a block of Consecutive Frames. In this case, the routing process proceeds as in step 4.
 - 1: if the reserved buffer in the Destination ECU is full and the Gateway must wait for a further Flow Control Frame. In this case, the process proceeds as in step 2.
 - 2: if the Destination ECU has currently no free memory for further blocks and some timing parameters cannot be hold any more. In this case, the routing process will be aborted and no further steps are necessary.
4. The gateway routes a saved block of CAN Consecutive Frames to the Destination Network before the timer N_Cr_Target expires. The number of block's frames and the time gap between them must be as specified in the BS and STmin parameters of the received Flow Control Frame on the Destination Network. Consecutive Frames must have the content structure as specified in table 3.3
5. The process proceeds as in steps 2-4 until the whole data has been routed.

Basically, the gateway can route CAN frames to the Destination ECU once they have been received from the Source ECU. If any timer could not be hold during the routing, the routing process will be aborted and an error will be saved in the software of the gateway. Timing parameters (N_Bs_Source, N_Cr_Source, N_Bs_Target, N_Cr_Target) and Control parameters (BS, STmin) of the Source and Destination ECUs are mostly not the same because of the different capabilities of the ECUs and the diversity of configured communication buses.

3.4.2 Segmented CAN to FlexRay TP Routing with Normal Addressing

The second TP routing paradigm explained in this thesis is the segmented CAN to FlexRay TP Routing with Normal Addressing, which is depicted in fig. 3.5. In this paradigm, a Source ECU located on a CAN network (Source Network) establishes TP data routing over the gateway to a Destination ECU located on FlexRay network (Destination Network). Due to different communication protocols of the Source and Destination Networks (CAN and FlexRay), the gateway has to invoke a protocol change.

Two different types of Flow Control Frames on CAN and FlexRay networks are utilized in this paradigm. The first type on the CAN network is the same as described previously in table 3.1. However, the second type on the FlexRay network has the structure as in table 3.4. In table 3.4, the parameter DA (Destination Address) in Byte 0 and Byte 1 is used to hold the address of receiving FlexRay ECU. The address of sending FlexRay ECU SA (Source Address) is encoded in Byte 2 and Byte 3 of the FlexRay Flow Control Frame. Flow Status (FS) in the low nibble of Byte 4 is required to report the current status of receiving FlexRay ECU and can have one of the following values:

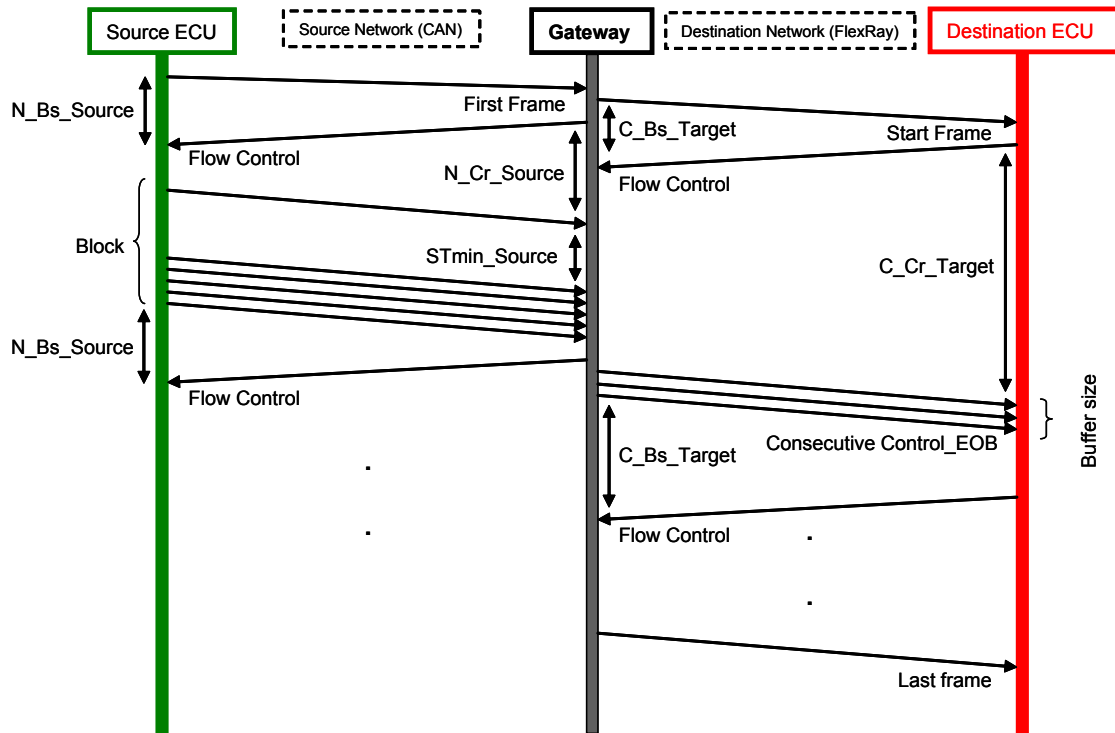


Figure 3.5: CAN to FlexRay segmented Transport Protocol Data Routing

byte 0	byte 1	byte 2	byte 3	Byte 4		Byte 5	Byte 6 + Byte 7	Byte 8 to Byte 41
15 ~ 0	15 ~ 0	7 ~ 4	3 ~ 0	BC	BfS	N/A		
DA	SA	8	FS					

Table 3.4: FlexRay Flow Control Frame Content Structure for Normal Addressing

- **3: Continue To Send (CTS)** indicates that receiving FlexRay ECU, which is the Destination ECU on the Destination Network in fig 3.5, is able to receive further data with a maximum as specified in the BfS and BC parameters.
- **5: Wait (WT)** indicates that sending ECU, which is the Gateway in this case, must wait for a further Flow Control Frame to proceed the routing.
- **6: Abort (ABT)** indicates that sending ECU, which is the Gateway in this case, must abort the routing process.
- **7: Overflow (OVFLW)** indicates that receiving FlexRay ECU, which is the Destination ECU on the Destination Network, has not enough memory to save the number of bytes specified and the Gateway must abort the routing process.
- **8-F: Reserved**

The parameter BC (Bandwidth Control) of a FlexRay Flow Control Frame is used to determine the receiving performance of the receiving FlexRay ECU to the sender, and the parameter BfS (Buffer Size) indicates the maximum number of data bytes allowed to be sent before waiting for a further Flow Control Frame to resume the routing.

Routing in this paradigm consists of multiple steps on both Source and Destination Networks as follows:

On the CAN Source Network

The steps on this side of TP routing process is the same as described in the previous TP routing paradigm on the Source Network.

On the FlexRay Destination Network

Once the Gateway was able to process the First Frame on the CAN Source Network, it implements the following steps on the FlexRay Destination Network:

1. The Gateway generates a FlexRay Start Frame with the content structure as described in table 3.5 and sends it directly without any delay to the FlexRay Destination Network. Byte 0 and Byte 1 of the FlexRay Start Frame hold the Destination Address (DA) of routing, whereas byte 2 and byte 3 are used to hold the Source Address (SA). The low nibble of Byte 4 will be 0 if the routing is acknowledged and 1 if it is unacknowledged. The parameter FPL (Frame Payload) defines the number of data bytes in the FlexRay frame, and the parameter ML (Message Length) defines the length in bytes of the whole data required to be routed in the current session. The data included in the FlexRay Start Frame is the same as in the received CAN First Frame on the Source Network.

byte 0	byte 1	byte 2	byte 3	Byte 4		Byte 5	Byte 6 + Byte 7	From Byte 8
15 ~ 0		15 ~ 0		7 ~ 4	3 ~ 0	FPL	ML	Data
DA		SA		4	0/1			

Table 3.5: FlexRay Start Frame Content Structure with Normal Addressing

2. The Gateway starts a new timer C_Bs_Target on the FlexRay Destination Network to receive a FlexRay Flow Control Frame from the Destination ECU.
3. The Gateway receives a FlexRay Flow Control Frame from the Destination ECU with the content structure as specified in table 3.4 before the timer C_Bs_Target expires. The value of parameter FS of received FlexRay Flow Control Frame depends on the current status of the Destination ECU and can be one of the following:
 - 3: if the Destination ECU is able to receive a maximum number of bytes as specified in BfS parameter. In this case, the routing process proceeds as in step 4.

- 5: if the reserved buffer of the Destination ECU is full and the Gateway must wait for a further FlexRay Flow Control Frame. In this case, the routing proceeds as in step 2.
 - 6: if the Destination ECU cannot continue the reception of data. In this case, the routing process will be aborted and no further steps will be necessary.
 - 7: if the Destination ECU has currently no free memory for further data. In this case, the routing process will be aborted and no further steps will be necessary.
4. Depending on the value of parameter BfS of received FlexRay Flow Control Frame from the Destination ECU, the Gateway routes data received from the Source Network to the Destination Network before the timer C_Cr_Target expires. The performance of routing is calculated based on the value of parameter BC of received FlexRay Flow Control Frame. Routing in this case proceeds as in one of following cases:

- If the value of parameter BfS is equal to 0, the Gateway routes the data using the content structure of Consecutive Frames as specified in table 3.6 without waiting for any further authorization from the Destination ECU and until the whole data has been routed. However, the last frame in this case has the content structure as defined in table 3.7.

In table 3.6, the parameter CFT (Consecutive Frame Type) is 5 if the Consecutive Frame is from type 1 and 6 if it is from type 2. The SN (Sequence Number) parameter of Consecutive Frames holds the order number and has a value between 0 and F. Once the value F has been reached, the next value for the next Consecutive Frame will be 0. The first Consecutive Frame after Start frame has the value of 1 that will be incremented by 1 for each further Consecutive Frame.

- If the value of parameter BfS is not equal to 0, the Gateway routes a block of bytes as specified in BfS parameter using the content structure of Consecutive Frames as specified in table 3.6. In this case, the last consecutive frame of the block must have the value 7 for CFT parameter. After that, the Gateway proceeds routing as in step 2. Here, the last frame of data must have the content structure as specified in table 3.7.

Basically, the gateway can route CAN frames once they are received from the Source ECU. If any timer could not be hold during the routing, the routing process will be aborted and an error will be saved in the software of the gateway. Timing parameters (N_Bs_Source, N_Cr_Source, N_Bs_Target, N_Cr_Target) of the Source and Destination ECUs are mostly not the same.

byte 0	byte 1	byte 2	byte 3	Byte 4		Byte 5	from Byte 6
15 ~ 0		15 ~ 0		7 ~ 4	3 ~ 0	FPL	Data
DA		SA		CFT	SN		

Table 3.6: FlexRay Consecutive Frame Content Structure with Normal Addressing

byte 0	byte 1	byte 2	byte 3	Byte 4		Byte 5	Byte 6 + Byte 7	from Byte 8
15 ~ 0		15 ~ 0		7 ~ 4	3 ~ 0	FPL	ML	Data
DA		SA		9	0			

Table 3.7: FlexRay Last Frame Content Structure with Normal Addressing

3.4.3 Other TP Routing Paradigms

Based on the complexity and the heterogeneity of configured networks of the AUTOSAR gateway and the TP configuration parameters of connected ECUs, diverse TP routing paradigms can be supported. The most known and common configured TP routing paradigms of an AUTOSAR gateway for CAN and FlexRay networks are listed below. The general concept of TP data routing shall be the same as described in the last two paradigm.

- unsegmented physical TP data routing from CAN network to CAN network with normal addressing (11 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to CAN network with normal addressing (11 bit CAN identifier)
- segmented physical TP data routing from CAN network to CAN network with normal addressing (11 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to CAN network with normal fixed addressing (29 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to CAN network with normal fixed addressing (29 bit CAN identifier)
- segmented physical TP data routing from CAN network to CAN network with normal fixed addressing (29 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to CAN network with extended addressing (29 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to CAN network with extended addressing (29 bit CAN identifier)
- segmented physical TP data routing from CAN network to CAN network with extended addressing (29 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to CAN network with mixed addressing (11 bit CAN identifier)
- segmented physical TP data routing from CAN network to CAN network with mixed addressing (11 bit CAN identifier)

- unsegmented physical TP data routing from CAN network to CAN network with mixed addressing (29 bit CAN identifier)
- segmented physical TP data routing from CAN network to CAN network with mixed addressing (29 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to FlexRay network with normal addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to FlexRay network with normal addressing (only unacknowledged, known message length with 11 bit CAN identifier)
- segmented physical TP data routing from CAN network to FlexRay network with normal addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to FlexRay network with normal fixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to FlexRay network with normal fixed addressing (only unacknowledged, known message length with 29 bit CAN identifier)
- segmented physical TP data routing from CAN network to FlexRay network with normal fixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to FlexRay network with extended addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented functional TP data routing from CAN network to FlexRay network with extended addressing (only unacknowledged, known message length with 29 bit CAN identifier)
- segmented physical TP data routing from CAN network to FlexRay network with extended addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented physical TP data routing from CAN network to FlexRay network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- segmented physical TP data routing from CAN network to FlexRay network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)

- unsegmented physical TP data routing from CAN network to FlexRay network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- segmented physical TP data routing from CAN network to FlexRay network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented physical TP data routing from FlexRay network to CAN network with normal addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- unsegmented functional TP data routing from FlexRay network to CAN network with normal addressing (only unacknowledged, known message length with 11 bit CAN identifier)
- segmented physical TP data routing from FlexRay network to CAN network with normal addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- unsegmented physical TP data routing from FlexRay network to CAN network with normal fixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented functional TP data routing from FlexRay network to CAN network with normal fixed addressing (only unacknowledged, known message length with 29 bit CAN identifier)
- segmented physical TP data routing from FlexRay network to CAN network with normal fixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented physical TP data routing from FlexRay network to CAN network with extended addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented functional TP data routing from FlexRay network to CAN network with extended addressing (only unacknowledged, known message length with 29 bit CAN identifier)
- segmented physical TP data routing from FlexRay network to CAN network with extended addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- unsegmented physical TP data routing from FlexRay network to CAN network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)

- segmented physical TP data routing from FlexRay network to CAN network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 11 bit CAN identifier)
- unsegmented physical TP data routing from FlexRay network to CAN network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)
- segmented physical TP data routing from FlexRay network to CAN network with mixed addressing (acknowledged/unacknowledged, known/unknown message length with 29 bit CAN identifier)

For further communication busses like LIN, MOST and Ethernet, there are other TP routing paradigms that are not discussed in this thesis. However, the concept proposed for test case selection and generation remains the same.

3.5 TP Parallel Routing of an AUTOSAR Gateway

TP parallel routing is a feature of the AUTOSAR standard that allows gateways to handle more than one *TP_Routing_Channel* in parallel (simultaneously). That is, an AUTOSAR gateway can be configured to serve multiple *TP_Routing_Instances* in parallel in order to obtain a better utilization of the resources and reduce the processing time.

In AUTOSAR, the maximum number of *TP_Routing_Channels* handled in parallel is configured statically. The configuration is achieved by adjusting some parameters of the AUTOSAR basic software modules of the gateway. These configuration parameters are for example, TP buffers in the PDU Router module, receive and transmit channels in the CAN TP module, receive and transmit channels in the FlexRay TP module.

Basically, AUTOSAR does not specify a limit for the maximum number of supported parallel *TP_Routing_Channels*, since the performance and capabilities of the deployed hardware and software of a gateway can vary and are in continuous improvement, as for example the performance of deployed processor, the size of used memory and the intelligence of deployed software. Generally, the maximum number is a configuration parameter that should be adjusted based on the capability of the gateway.

In order to explain the aspects of TP parallel routing of an AUTOSAR gateway, consider the simple example of E/E system depicted in fig. 3.6. This system consists of a Gateway that connects four individual networks together (Network₁, Network₂, Network₃ and Network₄). ECUs of the system are distributed as follows:

- ECU₁ and ECU₂ are located on Network₂.
- ECU₃ is located on Network₃.
- ECU₄ and ECU₅ are located on Network₄.
- An External Diagnostic Device is connected to Network₁. This kind of devices is considered as an ECU that is temporarily available to access ECUs of the system.

In order to enable TP data routing between the External Diagnostic Device and ECUs of the system, the gateway has the following configured *TP_Routing_Channels*:

- *TP_Routing_Channel₁* to route TP data from the External Diagnostic Device to ECU₁
- *TP_Routing_Channel₂* to route TP data from the External Diagnostic Device to ECU₂
- *TP_Routing_Channel₃* to route TP data from the External Diagnostic Device to ECU₃
- *TP_Routing_Channel₄* to route TP data from the External Diagnostic Device to ECU₄
- *TP_Routing_Channel₅* to route TP data from the External Diagnostic Device to ECU₅

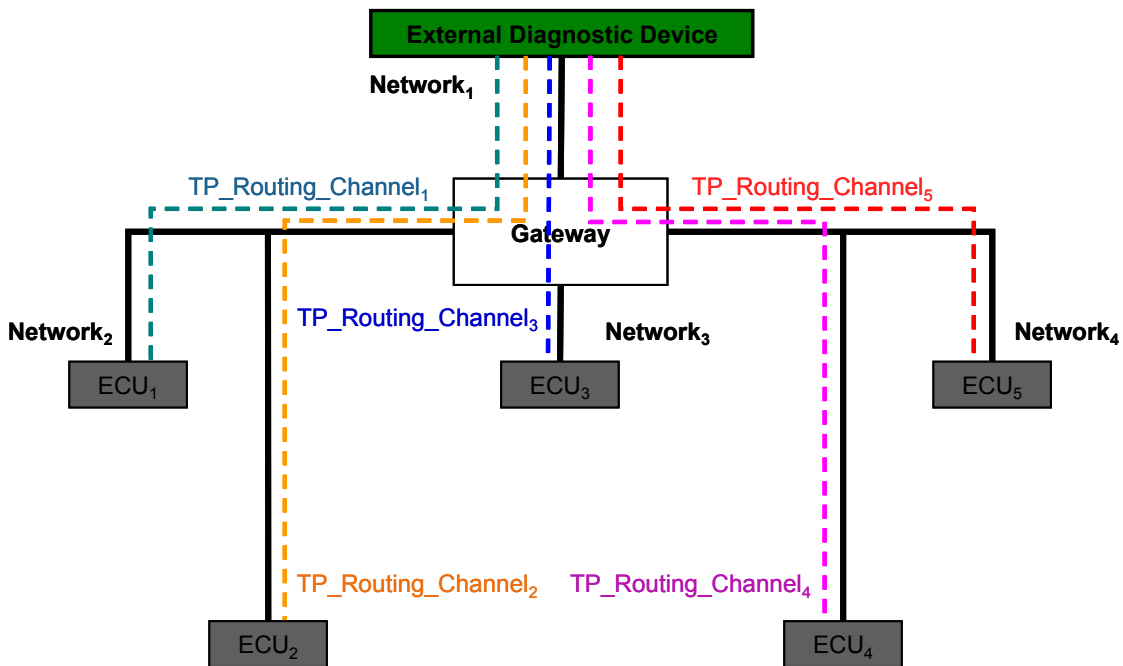


Figure 3.6: TP Parallel Routing Example of an AUTOSAR Gateway

Individual *TP_Routing_Channels* can be assigned to any of TP Routing Paradigms explained in section 3.4. Furthermore, employed parameters for TP data routing, such as BS, STmin and N_Bs for CAN TP routing, or BfS and Cycle Repetition for FlexRay TP routing, are ECU specific and have in general a heterogeneous nature. As mentioned before, the reason behind this heterogeneity is the different performance levels of corresponding ECUs and configured networks. In other words, the AUTOSAR gateway behaves differently during TP data routing if parameters or even connecting networks of communicating partners are different.

Let us assume in the example of fig. 3.6 that Network₁, Network₂ and Network₄ are CAN networks with a speed of 500 Kilobaud, and Network₃ is a FlexRay network with one channel and a speed of 10 Mbit/s. Additionally, assume that each individual *TP_Routing_Channel*

has configuration parameters different than the others. In this case, the behavior of the gateway would be different when TP data routing is established for example between the External Diagnostic Device and ECU₁ than between the External Diagnostic Device and ECU₃ because:

1. Network₂ as a destination network is involved in the first TP routing process, while Network₃ as a destination network is involved in the second TP routing. It is obvious that the combination (500 Kilobaud CAN, 500 Kilobaud CAN) of involved networks in the first TP routing process triggers another behavior of the gateway than the combination (500 Kilobaud CAN, 10MB FlexRay) in the second TP routing process.
2. Parameters of *TP_Routing_Channel₁* utilized in the first TP routing are different than parameters of *TP_Routing_Channel₃* utilized in the second TP routing. As mentioned previously, a *TP_Routing_Channel* comprises parameters required for TP routing. If the count, the type or the values of these parameters vary between two *TP_Routing_Channels*, the behavior of the gateway will be different once they are used to establish TP routing. In other words, parameters of *TP_Routing_Channel₁* causes a different behavior in the gateway than parameters of *TP_Routing_Channel₂*.

Another example of TP routing can be between the External Diagnostic Device and ECU₁, and between the External Diagnostic Device and ECU₂. Although the same source and destination networks are involved in this case, the behavior of the gateway will be different due to the different configuration parameters of *TP_Routing_Channel₁* and *TP_Routing_Channel₂*.

A modern External Diagnostic Device has the capability to establish parallel transmission of TP data and hence trigger TP parallel routing of the AUTOSAR gateway. In this use case, the gateway builds up individual *TP_Routing_Instances* simultaneously. Back to the example of fig. 3.6, let us assume that the gateway is configured to support four different *TP_Routing_Instances* in parallel. In this configuration possibility, the gateway would be able theoretically to establish TP parallel routing with one of the following combinations of *TP_Routing_Channels*:

- *TP_Routing_Channel₁*, *TP_Routing_Channel₂*, *TP_Routing_Channel₃* and *TP_Routing_Channel₄*
- *TP_Routing_Channel₁*, *TP_Routing_Channel₂*, *TP_Routing_Channel₃* and *TP_Routing_Channel₅*
- *TP_Routing_Channel₁*, *TP_Routing_Channel₃*, *TP_Routing_Channel₄* and *TP_Routing_Channel₅*
- *TP_Routing_Channel₂*, *TP_Routing_Channel₃*, *TP_Routing_Channel₄* and *TP_Routing_Channel₅*
- *TP_Routing_Channel₁*, *TP_Routing_Channel₂*, *TP_Routing_Channel₄* and *TP_Routing_Channel₅*

Each variant of these combinations has a different impact on the behavior of the gateway once they are used to practice TP parallel routing. For example, involving three CAN networks with 500 Kilobaud and one FlexRay network with 10MB as in the first combination leads to another behavior than involving three CAN networks with 500 Kilobaud as in the last combination. Furthermore, parameters of *TP_Routing_Channel₁*, *TP_Routing_Channel₂* and *TP_Routing_Channel₃* in combination with parameters of *TP_Routing_Channel₄* in the first combination trigger another behavior than in combination with parameters of *TP_Routing_Channel₅* in the second combination, and so on.

As AUTOSAR offers a particular grade of flexibility through various implementation and configuration variants, the complexity regarding configuration has been increased. Nowadays, it is challenging to find out the optimal configuration for an AUTOSAR based system, since the number of configuration parameters is very high and the interactions between these parameters require deeply understanding in diverse areas of software engineering. For instance, it is important to adjust the configuration parameters of an AUTOSAR gateway in order to enable a correct handling of TP parallel routing. On the one hand, the configuration has an important impact on the resource consumption of the gateway and, on the other hand, the configuration parameters of *TP_Routing_Channels* as well as their interactions affect the behavior to a large extent.

3.6 The Combinatorial Explosion Problem

In this section, the combinatorial explosion problem related to testing of TP parallel routing of an AUTOSAR gateway is discussed. The aim of testing is to verify the TP parallel routing functionality and measure its performance. Moreover, the gained information shall help in optimizing the configuration parameters of the gateway system.

The explosion problem is a well-known issue in software testing. Depending on the test object and the test methodology, the problem is referred to in different ways in literature. For example, the term "*state explosion*" or "*path explosion*" [17] is used to describe the explosion problem in model based testing where methods like symbolic model checking [18], partial order reduction model checking [19] [20] and symmetry reduction [21] are used to alleviate the problem. In the case of black box testing or functional testing [22], the problem is referred to as the combinatorial explosion which can be solved by means of combination test strategies [23].

In order to illustrate the combinatorial explosion problem of testing TP parallel routing of an AUTOSAR gateway, consider the E/E system example in fig. 3.7. The gateway in this example acts as a central point of communication between connected ECUs. To realize TP data routing, the gateway has n configured *TP_Routing_Channels* as follows:

- *TP_Routing_Channel₁* between ECU₁ and ECU₂
- *TP_Routing_Channel₂* between ECU₁ and ECU₃
- .
- .
- .
- *TP_Routing_Channel_n* between ECU₁ and ECU_k

Configured *TP_Routing_Channels* of the gateway define which ECUs are allowed to establish TP routing and determine, by means of configuration parameters, the behavior during the TP routing process.

Since AUTOSAR features parallelism for transport protocols [24] [25], the standard allows handling of multiple TP communication instances in parallel. That is, an AUTOSAR gateway can be configured to support a limited number of parallel *TP_Routing_Channels* among ECUs. For each supported parallel channel, the gateway shall reserve individual resources

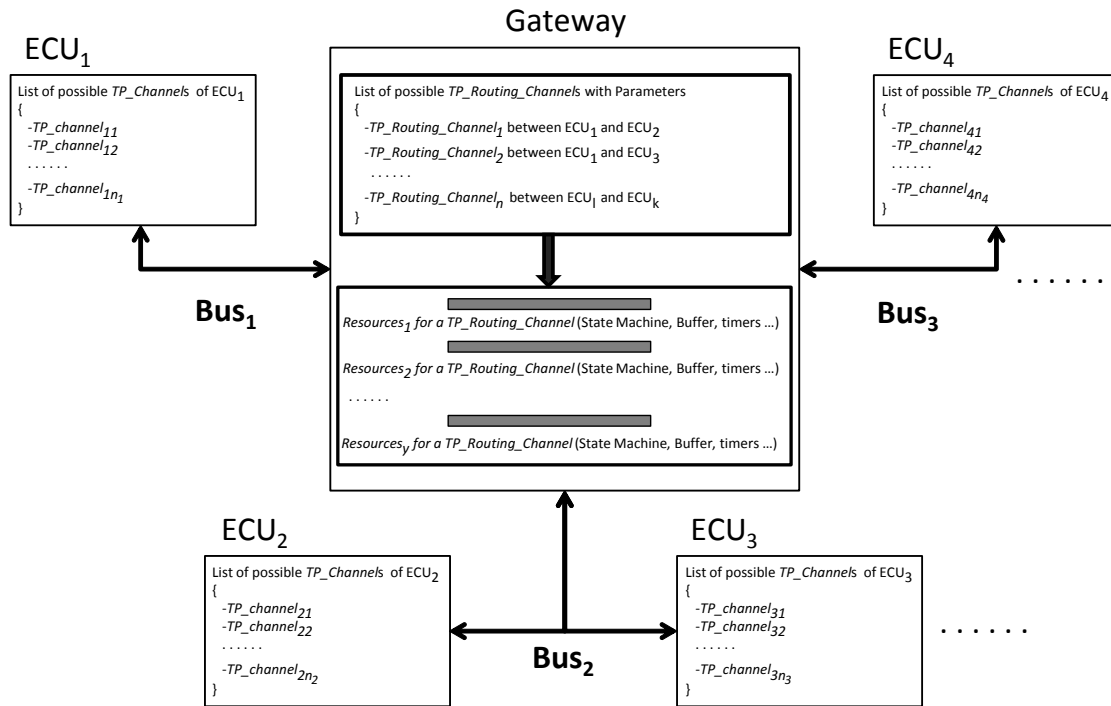


Figure 3.7: E/E System from the TP functionality Point of View

such as buffers, timers, variables and finite state machines. In order to support y parallel *TP_Routing_Channels*, the gateway has to reserve y individual instances of required resources as follows (see fig. 3.7):

- Resources₁ for the first *TP_Routing_Channel* of TP parallel routing
- Resources₂ for the second *TP_Routing_Channel* of TP parallel routing
-
-
-
- Resources_y for the yth *TP_Routing_Channel* of TP parallel routing

Reserved resources for parallel *TP_Routing_Channels* are generic and can be utilized to establish any of the configured *TP_Routing_Channels* among ECUs.

Generally, ECUs that have the capability to trigger TP routing over the gateway shall possess a specific number of configured *TP_Channels*, which are referred to as a list of *TP_Channels* under each ECU in the fig. 3.7. The number of configured *TP_Channels* can vary from one (as in most ECUs) up to hundreds (as in the External Diagnostic Device).

Once an authorized ECU (ECU that has a configured *TP_Channel*) triggers TP routing, the gateway applies related parameters of corresponding *TP_Routing_Channel* to one instance of the reserved generic resources. If a powerful ECU, like the External Diagnostic Device, or more

than one ECU trigger TP routing simultaneously, the gateway establishes TP parallel routing. In this case, the gateway applies related parameters of corresponding *TP_Routing_Channels* to the required generic resources.

However separate resources of the gateway, such as buffers and timers, are reserved for the channels of TP parallel routing, the main resources like the processor and communication busses are shared. For an optimum and correct utilization of the shared resources, suitable values for configuration parameters of *TP_Routing_Channels* of the gateway shall be chosen. Generally, the selection is a trade-off process that requires knowledge about the whole system. In this context, testing shall support the designer to verify a selected configuration or even to choose a suitable configuration variant for the whole system.

During the test of TP parallel routing of an AUTOSAR gateway, the combinatorial explosion problem arises. To explain the problem in this case, let us consider again the E/E system in fig. 3.7. The gateway is configured to serve "n" *TP_Routing_Channels* between connected ECUs. It is also configured to support "y" *TP_Routing_Channels* in parallel by means of reserved resources. System designers have to select suitable values for configuration parameters in order to implement the specification and meet performance requirement. Testing has the task to verify selected configuration parameters and to measure the performance. It has also the task of providing statistical information that can help to understand performance issues and optimize values of configuration parameters.

To verify the gateway's capability to serve "y" parallel *TP_Routing_Channels* correctly and determine the next value for "y" in case of errors, all possible combinations from 1 to y of *TP_Routing_Channels* should be included at least once in test cases. This results in a number of combinations to be tested which can be calculated using the equation (3.18).

$$X = \frac{n!}{y!(n-y)!} + \frac{n!}{(y-1)!(n-(y-1))!} + \dots + \frac{n!}{1!(n-1)!} \quad (3.18)$$

The equation(3.18) calculates the sum of all possible combinations for a number of (n) *TP_Routing_Channels*, where the selected number of *TP_Routing_Channels* in each term varies from y to 1.

The combinatorial explosion problem of functional testing of configurable concurrent systems like TP parallel routing of an AUTOSAR gateway, is different than described combinatorial explosion problem in literature. The combinatorial explosion problem mentioned in literature [26–28] shall be explained as in the following example:

Assume a distributed system that consists of a central unit interacting over communication channels with u units of the network: U_1, U_2, \dots, U_u . Each unit U_i uses a defined parameter p_i for communication. The parameter p_i shall have v_i possible configuration values. By assuming that configuration values of parameters are independent from each other, the number of possibilities in which the system can be configured would be $v_1 * v_2 * \dots * v_u$. If each possible configuration requires c test cases to verify it, the number of test cases for exhaustive test would be $c * v_1 * v_2 * \dots * v_u$. In a nontrivial software system, the values of u and v_i are large which leads to a huge number of possible combinations of parameter values.

Related to TP parallel routing, two main goals are defined for testing:

1. Measuring the performance of the gateway to handle parallel *TP_Routing_Connections* in a configured system.
2. Testing that the gateway is able to execute all supported paradigms of TP routing correctly.

In this case, the combinatorial explosion problem is more complex because:

- System input parameters are *TP_Routing_Channels* where each consists of a set of configuration parameters.
- The number of system input parameters is not fixed. It can be different for every new release of the system.
- System input parameters include also timing parameters where the interactions are difficult to resolve.
- The number of parallel *TP_Routing_Channels* “y”, which is also a configuration parameter, is used to build possible combinations to be tested. Combinations are any y elements of the system input parameter set. In case of errors, one of the goals is to determine the next “y” and verify it (performance measurement).
- In TP parallel routing, each additional instance will consume resources of the system and may lead to errors. Hence, it is not only a specific combination of *TP_Routing_Channels* which can affect the behavior and may reveal errors, but also the number of included *TP_Routing_Channels* and their values.

Software Testing

4.1 Introduction

Software testing is one of the main techniques of software Validation and Verification (V&V). It is defined as an activity, in which the software is analyzed to detect the discrepancies between existing and required conditions and to evaluate the features of the software item [29, 30].

Software Validation and Verification are software quality assurance activities used to ensure that a software system meets its requirements. According to IEEE [30], Verification is “the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase”. Verification activities may include technical reviews, walkthroughs, software inspections, traceability check, testing and audit. On the other hand, Validation is defined as “the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements” [30].

Before discussing software test techniques in details, some related definitions are stated [30].

- **Mistake:** a human action that produces an incorrect result.
- **Fault:** incorrect step, process, or data definition in a program.
- **Failure:** the inability of a system or component to perform its required function within the specified performance requirement.
- **Error:** the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
- **Specification:** a document that specifies in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristic of a system or component, and often the procedures for determining whether these provisions have been satisfied.

- **Informal specification:** statement about the properties of a product made using the grammar, syntax, and common definitions of a natural language.
- **Semi-formal specification:** uses combination of natural language with a notation with defined semantics to describe system specifications.
- **Formal specification:** uses mathematical notations such as logic and set theory to describe system specifications.

In general, there are diverse possibilities to classify software testing techniques. Most known classification paradigms are described in the next sections.

4.2 Testing Based on Actual Execution

According to this classification paradigm, testing is categorized in two major classes:

1. **Static Testing:** static testing [31] describes techniques for error detection which are used during the software development life cycle prior to application execution. Most static testing techniques, e.g., walkthroughs, reviews, inspections and data flow analysis, can be utilized to check any form of documents including source code, design documents, models, functional specification or requirement specification. These test techniques can only detect particular types of errors.
2. **Dynamic Testing:** dynamic testing [31] defines methods for testing through actual execution, i.e., with real data and under real or simulated circumstances. In order to implement dynamic methods, a working system, or a prototype of it, should be available. Therefore, dynamic testing cannot be used in the initial development phases. This type of testing requires some initial efforts to implement, but then, it can be re-used for various versions of the system or also various types of software.

4.3 Testing Based on Methodology

Two categories are available in this classification paradigm:

1. **Functional Testing (Black Box):** functional testing [32] (also referred to as “black box” testing) concentrates on the expected functional behavior of the system as specified in the functional requirements. In functional testing, the test object is being regarded as a black box and test cases are derived directly from the functional specifications without necessarily having to understand underlying details of the software design.
2. **Structural Testing (White Box):** structural testing [32] (also referred to as white-box testing) deals with the structure of the test object, i.e., it focuses on the form rather than the semantic of the functionality of the test object. Structural test techniques help in designing test cases based on the internal structure and design of the test object. Typical examples of this kind of testing are statement, branch, and path coverage on the basis of

control flow graphs. Another example is the data flow coverage on the basis of control flow charts.

4.4 Testing Based on Granularity Level

Three different granularity levels are available in this classification paradigm:

1. **Unit Testing:** unit testing refers to testing of separate system's units and it requires the low level design or code structure as a specification. Units can be a collection of procedures or functions performing a specific task of the test object. In unit testing, white box testing techniques are used to verify that the code does what it is intended to do at a very low structural level.
2. **Integration Testing:** integration testing is utilized to ensure the correctness of integrated system. It evaluates the interactions between software components. For integration testing, both black and white box testing techniques are used for the verification that is based on low and high level design specifications. Integration testing is often the most expensive and time consuming testing. Techniques for integration testing can include top-down integration, bottom-up integration or regression testing [33].
3. **System Testing:** based on high level design and requirement specifications, black box testing techniques are used to achieve system testing. In system testing, the complete system is exercised to ensure that all functions combine for the desired result. The main problem related to system testing is to figure out the cause of errors once they are uncovered.

4.5 Specification Based Testing

In this classification paradigm, testing techniques are categorized based on the type of specification used to describe the system. Specification based testing is mostly a black box testing that consists in deriving test data and constructing test cases from requirement specifications or test specifications of the test object. Specification based testing is independent from the implementation details of the system and can be launched before the system development begins. This is a very useful practice, since it opens the opportunity to develop tools and algorithms for automatic generation and execution of test cases regardless of implementation information.

Specifications used for testing can be generally classified in three different categories: informal specification, semi-formal specification and formal specification. Some common and often used instances of these categories are discussed below.

- **Written Document:** the requirements in this case are specified in a natural human language and fall in the category of informal specifications. Written documents are the oldest approach to express features and functionalities of the software system. However, they suffer from the following disadvantages:

- Lack of clarity and specificity.
- Ambiguity and incompleteness.
- Trying to include clarity in a written document could lead in complex systems to a document that is difficult to read.
- Functional and non-functional requirements are often mixed in written document, which results in so called requirements confusion where only one sentence can include more than one functionality and leads to requirements amalgamation.

To avoid these disadvantages and address the quality of written specification documents, IEEE published in 1998 a Recommended Practice for Software Requirement Specifications [34].

Once a software system specified in written documents has to be tested, the test engineer adopts written specifications concerned with the implementation to write a test specification document in a human language. Generally, a written test specification document describes how specified functions and interfaces must be tested in order to ensure their correct implementation.

In written document based testing, the quality of the test methodology depends strongly on the quality of written test specification document and hence on the experiences of the test engineer and his understanding of the functions to be implemented. Due to the manual definition of test specifications in this category, the complete possibilities of input and output variants are in most scenarios difficult to be determined. That is, the test engineer has to spend a lot of time to design and write input patterns in order to make it possible to generate test patterns automatically. Consequently, the effectiveness and efficiency of test patterns depend to a wide extent on the test engineer. In addition, the covering degree of the test is limited because of the very low automation level supported with written test specification documents.

In spite of all mentioned disadvantages, written documents are widely used because of their low cost and the possibility to apply them in most phases of the development process. Written test specification documents are typically very abstract and describe either the environment of the system or the interfaces; therefore they are generally suitable for Black-Box and Integration test and not for White-Box or Unit test. Furthermore, testing strategies developed from written documents are mostly simple strategies, which cover only simple scenarios, because of the high costs related to manual construction of test cases.

- **DTD and XML:** Document Type Definition (DTD) and Extensible Markup Language (XML) [35] are standards defined at the World Wide Web Consortium (W3C) [36]. The aim of the standards is to determine general rules and constraints for structuring flexible, human and machine readable files. In consequence, the standards define organizing elements to encode information in a regular way into a documents.

Necessitate of such standards arises when software applications developed by different companies with diverse purposes require to communicate with other entities (such as other

software applications or developers). In this case, the exchanging data must be understandable for all communication partners. To realize a mechanism for that purpose, XML and DTD were introduced as standardized formats to represent exchange data. In simple words, XML and DTD are communication languages between software applications, or some times between developers and software applications, that are working independent and conceivably on different platforms.

XML and DTD are used mainly to describe the external interfaces of entities that could be in this context a software application or a software module. That is, they cannot be used to describe the internal behavior of specified entities; therefore, they fall in the category of semi-formal specifications.

A special case of this category of semi-formal specifications is the AUTOSAR Extensible Markup Language (ARXML) [37] developed for automotive field. The concept is the same as in XML but the rules and the end users are different.

As XML and DTD are used to specify the interfaces of entities and not the internal behavior, testing based on this kind of specifications can be only a Black Box testing.

Generally, testing the functionality of interfaces based on DTD or XML specifications is simple, since input and output data of the interfaces are mostly part of the specification. The input interfaces can be tested with data included in the DTD or XML instances, and the output interfaces can be verified by testing the conformance of the output DTD or XML instances generated by the output interfaces.

One of the most benefits of this kind of specification is the ability to automatically interpret included information in clearly defined rules. In contrast to written documents, DTD and XML standards enable individual functionalities of the system to be encoded clearly and separately in the specification file. Nonetheless, testing based on DTD and XML documents suffers from following disadvantages:

- DTD and XML are open standards with a certain level of freedom for designing specification documents; therefore the document itself could be erroneous and must be validated against the intended schema to ensure its conformance. Schema conformance check includes testing if the document satisfies demanded rules and constraints of used schema. Several aspects must be validated in this context to assure the quality of the document [38]. Syntactical testing is for example a widespread approach. Moreover, testing the semantics is also available in literature [39]. Conformance test is an extra overhead added to this category of specification based testing.
- Different releases of the standard lead to the problem of managing the differences and the compatibility between the different releases. Some applications could understand one schema but have problems to understand another one.
- This kind of specification based testing is restricted to Black Box testing, as the data representation includes only the communication data, which are the inputs or outputs of communicating partners. Details about the internal functionality of the application cannot be included.

- Tools for generating the test instances (input instances of specific programs) based on a specific schema are also available, but only for applications using the same kind of specification language [40] [41] [42].
 - DTD- and XML-based specifications are until now not able to describe the internal behavior of a function. They are also not able to describe complex state machines or sequence diagrams and therefore could not be used for White Box testing.
- **Algebraic/Co-algebraic Specification:** Algebraic specification emerged in the 1970's as a formal specification method of abstract data types [43]. The technique of algebraic relies on the observation that programs are mostly modeled as a collection of functions, modules or classes which handle sets of data values by using mathematical operations [43]. Algebraic specification is also called property based specification [44]. They are particularly well-suited for the specification of interfaces. This comes from the fact that interfaces are defined as abstract data types or object classes. However, in the case where object operations are not independent of object states, this kind of specifications has problems to specify systems.

For dealing with the reactive behavior of systems represented as automata, the co-algebraic specification has been introduced [45]. Through final co-algebras, which are the key elements of co-algebraic specification, infinite set of system behavior can be described.

Testing based on algebraic and co-algebraic specification is mainly a Black Box testing, in which test cases are derived from system algebraic and respectively co-algebraic specification [46] [47]. Generally, algebraic based testing has some known problems such as the oracle problem and the infinite test data sets. The oracle problem describes the case where it is not possible to decide if an execution returns a correct or a faulty result. This comes from the abstract nature of the functional specifications, since data types are represented in the specification in an abstract form. However, through the execution of the program, the data types of results are concrete.

Two issues are raised with the oracle problem. The first issue is that the expected results are not known in advance for some input values and hence cannot be included in the specification. The second issue arises when a comparison needs to take place between expected values defined in the specification and returned values from exercising the SUT.

The comparison issue is straightforward if the two values are from predefined sorts of programming languages such as *integer* and *boolean*, which are also referred to as *observables*. The problem in this case can be solved by using the equality operation supported by the programming language. However, if the results returned from system execution and results in the specification are from the type *user-defined*, which is also referred to as *non-observables*, the comparison problem would be more complex to solve [48]. To deal with the comparison problem, some hypotheses can be defined and utilized [49].

Related to the problematic of infinite test data sets, the issue can be handled by choosing a selection criteria based on some known hypotheses such as regularity and uniformity [48].

Generally, all algebraic specification based test techniques introduced in literature were concerned mainly with the test data selection phase, which is understandable, since the

most problems are related to this phase of test. There is a small research on techniques of test case generation based on algebraic specification. Some applications found in literature are: unit test of object oriented programs [50], test of object oriented programs [51] and test of java components [52].

Since algebraic and respectively co-algebraic specifications present only functional properties of the system, they are not used to deal with testing of non-functional aspects such as performance and robustness. The well-known algebraic and co-algebraic specification languages supported through tools are CASL, COCASL, HASCASL and HETCASL.

- **Model Specification:** Models are formal specifications that describe the SUT on a specific abstraction level. Model-based testing (MBT) uses formal models of software systems to derive test cases. In general, two types of models can be identified in MBT. The first model is the finite state machine (FSM) [53] and the second is the labeled transition system (LTS) [54].

In MBT which utilizes FSM, a structural coverage criterion like transition coverage, state coverage or path coverage is used to derive test sequences from the model of the FSM [55]. The fact that most approaches to MBT based on FSMs deal only with deterministic FSMs, leads to a restriction of the method, as testing of reactive systems. To optimize the number of tests abstracted from FSM with respect to length, overlap and other goals, many research works have been achieved in the past (see Transition-Tour [56] and Unique-Input-Output [57]).

Generally, FSMs are not expressive enough to model real software systems. Therefore, extended finite state machines (EFSM) have been proposed. These extend the regular FSMs with data state variables and data parameters for inputs and outputs. A typical example of EFSMs is a state chart.

In MBT that is based on LTS, a conformance relation is defined to check the conformance of a SUT with respect to a LTS model. A well-known implementation of LTS in MBT is Input/Output Conformance (IOCO) [58] with numerous extensions for real time systems [59] [60] and for symbolic LTS [61] [62]. Another implementation is the alternating simulation [63] in the framework of Interface Automata (IA) [64].

As LTS based MBT describes only a conformance relation, test selection strategies have been developed for this type of models (see for instance selection strategies based on coverage criteria [65], on metrics [66], on test purposes [67], on state partitioning [68], on graph traversal [69] and on model slicing [70]).

In general, two main challenges are a continuous line of research in MBT. The first challenge is related to test data generation from the system model, where various approaches are tried [71, 72]. However, the second challenge is concerned with testing real-time systems, where the test generation algorithms should determine when an input should be sent and when an output should be received [73].

In respect to modeling notations, there are a wide variety of used notations. Notations can be generally either textual or graphical (as in UML). ETSI produced in 2011 a standard

that collected a number of tool-independent general requirements on notations for MBT [74].

4.6 Random Testing

Random testing (RT) is a fundamental software testing technique, in which test cases or test data are generated in a random manner. RT may be in some cases the only possible testing method if the specifications are incomplete or unavailable. However, it has been often proven that the effectiveness of random testing in detecting software failures is controversial [75]. Therefore, Cohen et al. [76] proposed the adaptive random testing (ART) in order to enhance the effectiveness of failure detection of RT. The concept of ART is based on the observation that input data which is able to reveal failures in the test object, is frequently clustered into contiguous failure regions [77]. That is, if a run test case has not revealed a failure, the next test case should be far from the already executed one. As a result, test cases should be distributed over the input domain. This concept has been implemented in various algorithms [78] [79] [80].

Anti-random testing [81] is another variation of random testing that improves the fault-detection capability by employing the location information of previously executed test cases. In contrast to ART, anti-random testing is deterministic. However, the first test case is selected randomly.

4.7 Search Based Testing

Search based testing is another category of software testing. Its focus is the optimization of algorithms that search for test data and/or test cases. Tests generated by search based algorithms aim to maximize the possibility of finding errors while minimizing the costs. The key issue of search based techniques is the design of a so-called fitness function that represents the desired test objective. During search, the fitness function is used to guide search algorithms towards the represented test objective. Test objectives can be any of the known goals of testing such as functional [82], structural [83], non-functional [84] or state-based [85].

Combinatorial Testing

In this chapter, the state of the art in Combinatorial Testing (CT) or Combinatorial Interaction Testing (CIT) is introduced and discussed. Moreover, a modified and extended combinatorial test process is proposed, which adopts recent research ideas and experiments of the last years. The extended combinatorial test process is applied in chapter 6 to solve the combinatorial explosion problem related to testing of TP parallel routing of an AUTOSAR gateway and measuring its performance.

5.1 Introduction

CT is a branch of software testing that consists of various active research areas. These areas are the key issues used to build the combinatorial test process proposed in this thesis (see fig. 5.1).

CT has been given an increasing attention in the last decade as modern software systems tended to be more flexible and highly configurable in order to support various platforms and environments. This trend of software systems introduced a new grade of complexity in terms of design, implementation, size, number of inputs and configuration possibilities. As a result, too many possible combinations of input parameters or software configuration variants shall be tested once the system has to be verified. Generally, three different goals can be recognized while testing of such software systems:

- The system configuration variant is considered to be correct and testing has to verify the software implementation (implementation testing)
- The software implementation is considered to be correct and testing has to verify configuration variants (configuration testing)
- Testing has to find a trade-off between software implementation and system configuration variants (trade-off testing)

In software testing, input parameters of the SUT and their interactions must be considered for the test to ensure the accurate detection of different software bugs. However, in complex software systems, like industrial applications or highly configurable applications, this leads to a large number of possible test cases. This problem is referred to in literature as the Combinatorial Explosion Problem [27] (see 3.6 for more details about the combinatorial explosion problem related to testing of TP parallel routing of an AUTOSAR gateway).

Generally, the consideration of all input parameters or configuration parameters and their interactions in exhaustive testing is impossible due to time and resource limitations [86]. Therefore, innovative and systematic approaches are required to ensure an efficient and sufficient coverage of interactions with a minimum number of test cases. To achieve this goal, strategies of CT have been developed [23].

Strategies of CT are approaches to functional testing or black-box testing. They are used to select specific test cases by combining test input parameters to detect interaction errors with an acceptable number of test cases [87]. Earlier approaches such as the category partition method [22], equivalence partitioning [75] or boundary analysis [87], tried to reduce the number of test cases by splitting input parameters of the test object into subsets whose parameters trigger similar behavior in the test object. Subsequently, representatives are chosen from the subsets to construct test cases. Sampling with the help of representatives contributes to the reduction of test cases. However, it is not appropriate to effectively tackle bugs due to interactions. To cover interactions between input parameters, combination strategies have been developed. Currently, a wide range of combinatorial approaches are existed to close this gap in software testing such as AETG [26], IPO [88], TCG [89] and many more.

To apply CT, a test process is required. Some combinatorial test processes have been already suggested in literature [90]. However, refinements and adaptations are needed to follow research ideas and experiments developed over the last years. Figure 5.1 shows a modified and extended test process suggested in this thesis to apply CT. The test process consists of multiple activities that represent the modern key issues of CT. These steps are explained in details in the next subsections.

5.2 Input Parameter Model (IPM)

Designing of an IPM is the first activity in the proposed CT process. This activity has to be performed carefully, since the structure of the resulting IPM may affect the effectiveness and the efficiency of the testing process.

An IPM can be defined as a possible structure of system parameters with associated values on a particular abstraction level. Commonly, designing of IPM is a creative process that depends on the skills of the test engineer and his understanding of the system's functionalities. Therefore, the same SUT can result in different IPMs.

To guarantee consistent and coherent IPMs in CT, formal specification may be used. However, most industrial and commercial software systems suffer from the absence of formal specification.

According to the proposed test process in fig. 5.1, the design process of IPM comprises various steps which are explained in the next subsections.

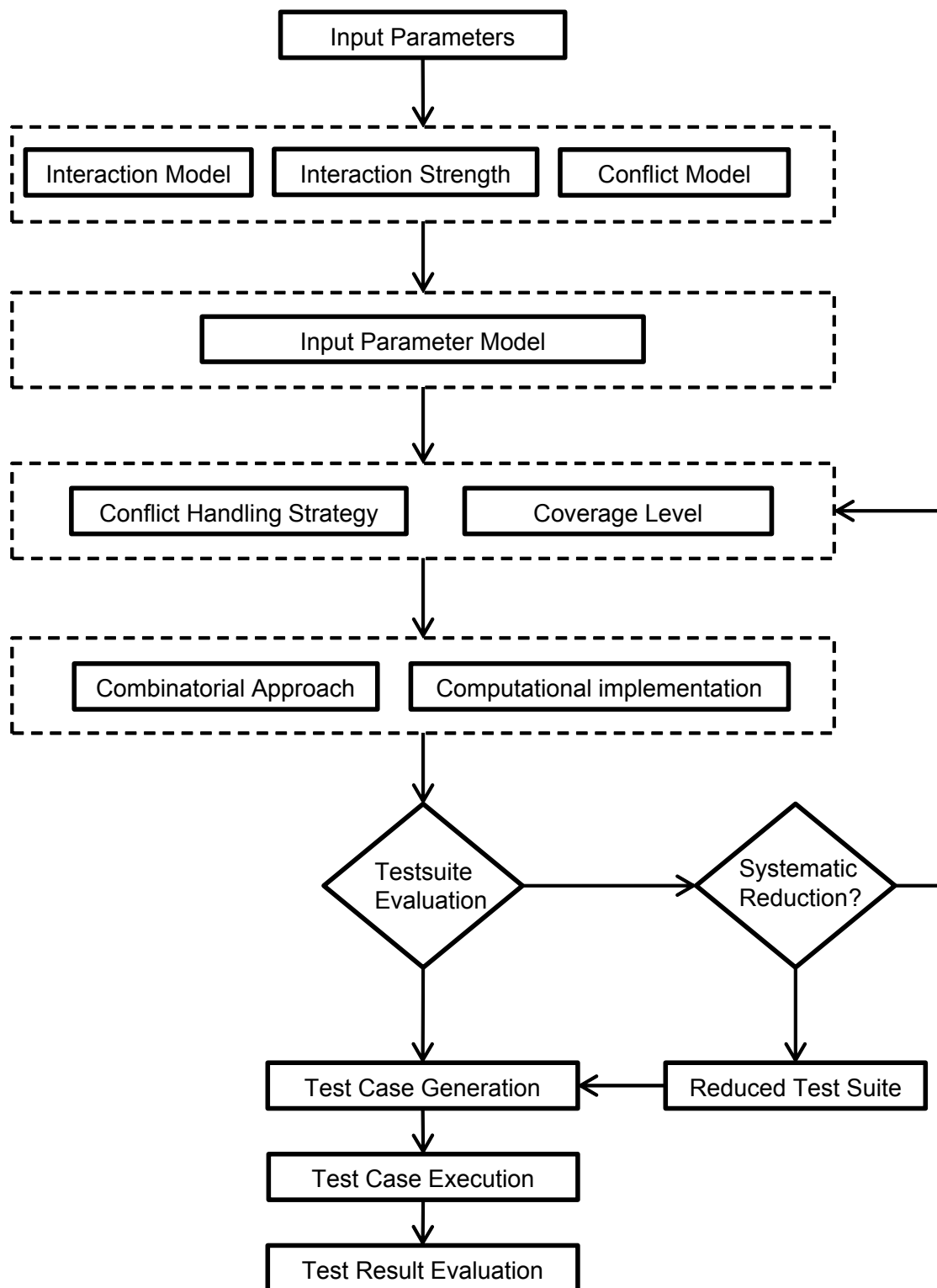


Figure 5.1: Extended Combinatorial Test Process

5.2.1 Interaction Model

In this step of the IPM design process, the test engineer has to determine and define the Interaction Model of input parameters based on the system specification. The SUT can be often represented either by means of input parameters or input/output parameter relationships. The representation option depends on the system specification and the possibility to determine output parameter values in relation with input parameter values. It is important to mention here that in most complex systems the input-output relationships are hard to resolve (see the oracle problem [91]).

Since CT have been emerged to fight the Combinatorial Explosion Problem while covering parameter's interactions, the model of parameter interactions may contribute to this goal [92]. According to the research work in the area of interaction models, there are three possible parameter interaction models that can be classified for CT.

1. Pure input parameter interaction model: this interaction model supports the generation of input parameter combinations without considering any relationships to the output parameters.
2. Input-output parameter interaction model: this interaction model supports the generation of input parameter combinations by considering the input-output relationships of the system parameters.
3. Mixed interaction model: in this interaction model, some of input parameter combinations are generated without taking into consideration any relationships to the output parameters, while others with the help of input-output relationships.

5.2.2 Interaction Strength

As well as determining the interaction model of system input parameters, the test engineer has to define the interaction strength of these parameters. Two cases are noted in the literature regarding the strength of parameter interactions:

1. Uniform Strength Interaction: here, all input parameters are assumed to be uniformly interacting (i.e. with constant interaction strength (t) throughout). To test all interacting parameters, the test suite must cover all the t -way combinations at least once [93], i.e., for 3-wise interaction testing, each exhaustive parameter-value combination of 3-way interacting parameter tuples must be tested at least once.
2. Variable Strength Interaction: in this case, different subsets of input parameters can have different interaction dependency [94].

5.2.3 Conflict Model

Most approaches to CT dictate that system input parameters must be independent of each other. However, experiences have often shown that some system input parameters may be mutually

exclusive [95]. As a result, invalid test cases can be generated. These must be removed in order to produce a valid test suite.

In CT, system input parameters that are mutually exclusive, have relationships called conflicts. A conflict is an impossible combination of some values of two or more parameters. It could be also an impossible combination of any values of two or more parameters.

A conflict can be explained with the help of a simple example. Assume a system that can be delivered in three different variants: low-, middle- and high-end. The operating function of the system can be either from the type basic or comfort for the middle- and high-end variants. However, the low-end variant can be only delivered with the basic operating function. In this example, the system can be described with the help of two variables. The first variable (*variant*) can have one of the possible values *low*, *middle* and *high*, and is used to describe the variant. The second variable (*function*) represents the operating function of the system and can have one of the values *basic* and *comfort*. It is obvious that the combination of the two parameter values (*low*, *comfort*) is conflicting, since the low-end can only be delivered with the basic operating function.

In order to generate a valid test suite, conflicts among system input parameters must be resolved. In a complex system, the number of conflicts can be large [95]. Therefore, they must be modeled in a consistent and effective way. Generally, constraints or conditions are the typical methods to model conflicts. It is the responsibility of the test engineer to design a good Conflict Model in this step, since designing of good conditions or constraints to describe conflicts may affect the test automation process and have an impact on the selection of conflict handling strategies explained later in 5.4.

5.3 Coverage Level

In this step, the test coverage level is selected and designed. The test coverage level is a metric that measures the percentage of covered parameter value combinations relative to the total combinations. It is used to define the degree of interaction required to get confidence in the SUT. Additionally, it can be used to evaluate the effectiveness and the efficiency of a test suite.

Defined coverage level is a key factor that may affect the selection process of combinatorial approach, i.e., it may force the usage of specific combinatorial approach or combination of multiple approaches. In general, the coverage level is referred to in literature as t -way, where t can take a value between 2 and the maximum number of system parameters N . Since 1-way coverage level demands that every value of every parameter is included at least once in a test case, the interactions between parameter values are not the focus of 1-way coverage level. The lowest coverage level in CT is 2-way. It demands that every possible pairs of parameter values are included at least once in a test case. The extreme case is N -way coverage level, also called exhaustive coverage. With the N -way coverage level, every possible combination of parameter values is required to be included at least once in the test suite.

To explain the various coverage levels and their impact on the test suite, consider the example of a system with 10 parameters, where each parameter has 5 different values. Depending on the selected coverage level, the test suite may have in this case, 1,125 2-way combinations, 15,000 3-way combinations, 131,250 4-way combinations, 787,500 5-way combinations and so on. The

N -way coverage level or the exhaustive test in this case is the 10-way, where the test suite would have 9,765,625 possible parameter value combinations.

Kuhn et al. [96] [97] studied the faults in various software systems and found that all known faults are caused by the interactions among 6 or fewer parameter values. This can be true, when the interactions are not the main source of errors. However, in complex systems with high interaction relationships, the defined coverage level will have a high impact on the quality of testing. The same issue can be recognized while verifying the performance of SUT, where every higher interaction level leads to another behavior and reveals another performance symptoms. For that reason, the coverage level must be clearly and reasoning defined, as it has a direct impact on the efficiency of the test suite and the effectiveness of fault detection or performance aspects.

While defining a suitable coverage level, the interaction strength among parameter values must be examined. The aim is to find out if parameter values have a fix or a variable strength of interaction among each other [98]. In general, two types of interaction strength among system input parameters can be stated, fix and variable. With the fix interaction strength, all parameter values are known to have the same strength of relationship to each other. In contrast to fix interaction strength, parameter values have different strength of relationships to each other in the variable interaction strength. That is, one coverage level is defined for systems with fix interaction strength while various coverage levels can be defined for systems with variable interaction strength.

Another aspect related to coverage levels is the effect of conflicts between parameter values. Handling conflicts in the IPM can cause a loss of the defined coverage level. It is the task of the generation algorithm to keep satisfying the defined coverage level after handling conflicts in the IPM.

5.4 Conflict Handling Strategy

To avoid invalid combinations of system input parameter values and hence invalid test cases, impossible combinations must be prohibited with the help of a conflict handling strategy in this step. In complex systems with large number of input parameters and conflicts, automatic detection and removal/avoidance are decidable for an effective test case selection and generation methodology. A general factor of a feasible conflict handling method is the observance of the selected coverage criterion. That is, after removing or avoiding invalid combinations, the test suite must still be complied with the defined coverage level.

Two methods to handle conflicts in IPMs are widely mentioned and used in literature. The first method "*sub-models*" [26] [99] handles conflicts by splitting the IPM into multiple conflict-free sub-IPMs. To achieve that, parameters involved in conflicts are determined. This is achieved by the definition of a Conflict Model as mentioned previously. Subsequently, the parameter with the least number of values is selected to be used for the splitting process. The splitting process creates sub-IPMs, where each one contains one value of the selected parameter along with all the values of every other parameter. Resulting sub-IPMs that contain conflicts involving the value of the selected parameter are processed then by omitting values of the other parameters that cause the conflicts. The method is applied recursively to the resulting sub-IPMs if other conflicts still remain. If all final sub-IPMs are conflict-free, the possibility to merge some of them shall be

examined. Finally, test cases are generated from all sub-IPMs separately and the final test suite is the sum of all generated test cases from all sub-IPMs.

The second well-known method to handle conflicts is the "*avoid*" method [26]. This method is simply prevents the selection of conflicting combinations of input parameter values while trying to satisfy the defined coverage level.

Further methods have been also proposed by Grindal et. al [100]. For instance, the *abstract parameter* method, in which two or more parameters involved in a conflict are represented by means of an abstract parameter whose values are only conflict-free sub-combinations of the represented parameters. Another example is the *replace* method, in which conflicts are handled after generating the test suite by replacing the conflicting values of parameters with an arbitrary values that are conflict-free.

5.5 Combinatorial Approach

The combinatorial approach is one of the most important key issues of the CT process and has to be selected in this step. It is the technique used to select test cases by sampling subsets of the input space of SUT in a systematic way. The aim of sampling is to solve the Combinatorial Explosion Problem caused by the large interaction space of system parameter values. Generally, the sampling process relies on a constructed IPM that represents the system as parameters with possible values (see IPM 5.2). From the IPM, combinatorial approaches select combinations from the Cartesian Product of the system parameter values. Thereby, the focus is to detect interaction errors caused by possible combinations.

The first utilization of combinatorial approaches to software testing goes back to the year 1985, where Mandl used the concept of Orthogonal Latin Squares [27] to sample 2-way combinations of parameter values to test the Ada compiler. In 1988, Ostrand and Balcer [22] developed the Category Partition (CP) method, in which the system is modeled as categories and partitions, each with a set of choices that represent equivalence classes for testing. The focus in the CP method was to reduce the input space for testing by selecting representatives from equivalence classes. The method built later the basis for the development of combinatorial approaches as an important area of software testing.

Since the interactions between parameter values are not the concern of CP, the method does not recognize the Combinatorial Explosion Problem caused by parameter value interactions. Therefore, Brownlie et al. [101] presented in 1992 an extension of Mandl's concept, called the Orthogonal Array Testing System (OATS), to test a weather forecast generator. The essential key of the OATS is that all pairs of input parameter values are tested exactly once. The test suite resulted by applying OATS was able to detect errors that had never been detected before. However, applying OATS in complex systems still results in a large test suite. In 1996 [102] and 1997 [26], Cohen et al. presented the use of Covering Arrays (CA), motivated by the idea that all pairs must be tested at least once instead of exactly once.

Previously mentioned approaches to CT share the same assumption that all input parameter values interact in an uniform manner. In 2004, Cohen observed that the strength of interaction may be not uniform among all parameter values. Therefore, he proposed the variable strength interaction approach [98]. The approach is based on the observation that in most real systems,

some parameter values exhibit stronger interaction level than the others. In this case, it is useful to separate the input parameters in various sets and apply different coverage levels based on the strength of interaction. This is more natural and practical, especially in complex systems with a large input space. However, the concept of variable strength interaction requires knowledge about the SUT to determine sets of parameter values and their strength of interaction. This makes the variable strength interaction approach lose one of the advantages of CT, i.e., the test engineer must have the required knowledge to form sets of parameters and define the corresponding coverage levels.

Generally, combinatorial approaches utilized to construct test suites from IPM can be classified into four main categories: greedy approaches, algebraic approaches, random approaches and heuristic search approaches.

In greedy approaches, there are two different techniques to select parameter value combinations. In the first technique, combinations are constructed in a successive way such that each subsequent combination covers as many uncovered interactions as possible until the test suite has been completed. Examples of this technique are AETG [26], PICT [103] and density-based greedy algorithm [104]. In the second technique, all t -sets for the first t factors are generated and then incrementally expands using heuristics until the coverage level has been reached. Examples of this technique can be found in [86] and [105]. Greedy approaches have the advantage that they are speed in generating the test suite. Therefore, they are the most used approaches for test suite generation in CT.

The second category of combinatorial approaches is called algebraic approaches. Its main advantage is the ability to efficiently construct test suits in a short time. However, with algebraic approaches, it is difficult to produce accurate results on a board and general variety of inputs. Latin squares [27], orthogonal arrays [101] and sequence covering arrays [106] are examples of algebraic techniques for combinatorial combination selection and test case generation.

In the third category of combinatorial approaches which is referred to as random algorithms, combinations of parameter values are selected randomly based on a distribution function. It is obvious here that the effectiveness of this class of approaches depends to a wide extend on the quality of the distribution function and the distribution of possibly available errors.

The last category of combinatorial approaches utilizes techniques of heuristic search and artificial intelligence to sample combinations of system parameter values. Examples in this category are simulated annealing [107], genetic algorithms [108] and ant colony algorithms [109]. The main advantage of combinatorial approaches from the heuristic and artificial intelligence areas is the ability to produce a small test suite. However, these approaches have a negative side. They consume a high duration time to compute and select combinations.

5.6 Computational Implementation

Since finding an optimal test suit with less number of test cases has been proven to be an NP-complete problem, the computation process of combinations can be labor-intensive and time-consuming. This can be seen in algorithms with high processing time such as genetic algorithms [108] and ant colony algorithms [109], or in complex systems with large input space, complex

parameter relationships and complex constraints. Furthermore, a high coverage level ($t > 6$) will make the computation aspect an important issue [110].

Othman et al. [92] gave attention to the computational problem as key issue of CT. In their survey, they suggested that the computation can be also implemented with the help of parallel computing. In contrast to sequential computing used for the implementation in most current combinatorial algorithms, parallel computing aids in optimizing the computational time and performing higher coverage levels. Depending on the nature of computation algorithm, parallel computing may have no contribution to optimizing the computation, as for instance, the algorithms with recursive generation. Therefore, the introduction of parallel computing in CT is feasible only for some approaches and can be seen as an optional step in the proposed process for CT.

5.7 Test suite Evaluation and Systematic Reduction

In this step, the test engineer provides a mechanism to evaluate the generated test suite. The aim is to examine the validity of the test suite with respect to defined coverage level and the absence of conflicts as well as the number of test cases. In the case that the test suite remained large, a decision shall be met. Either a different coverage level shall be selected, or a systematic reduction approach shall be developed. A reduction approach in this phase of the CT process is mostly intuitive and system specific.

5.8 Test Case Generation, Test Case Execution and Test Result Evaluation

Using formal specifications to describe the test requirements is of vital importance to the automation of the test process in these steps. For the test case generation, the availability of formal specifications supports an automatic transformation of abstract test cases into executable test cases. The transformation procedure is responsible for creating real test case inputs and identifying the expected test case outputs. Additionally, complementary information about the communication interface with the SUT shall be added by the transformation procedure. Subsequently, executable test cases are run and the results are recorded for evaluation.

Verification & Performance Measurement of Transport Protocol Parallel Routing

In this chapter, a methodology is proposed to apply the combinatorial test process explained previously in fig. 5.1. The methodology aims to verify TP parallel routing functionality of an AUTOSAR gateway system while measuring its performance. Several goals are in the focus of the methodology:

1. Building a creative IPM that helps to fight the combinatorial explosion problem raised during the verification and performance measurement process.
2. Handling conflicts in the IPM.
3. Defining a coverage level that satisfies the verification objective.
4. Reducing the size of resulting test suite systematically.
5. Automating the test process.

As explained previously in chapter 3, a big number of configuration variants is possible for a complex AUTOSAR system like the central gateway. Therefore, the implementation of the combinatorial test process in this thesis focuses only on one possible configuration of the system, i.e., once the system has been configured, the methodology can be applied.

Since the development of modern complex and highly configurable systems is an expensive process that is carried out in multiple phases (see V-Model in fig. 6.1), various types of errors can be introduced in the system. This thesis is concentrated on covering errors of TP parallel routing which are provoked by combinations of input parameters under a predefined configuration variant of the system. This kind of testing, called System Testing in fig. 6.1, covers in this case the following errors:

- Functional errors which could not be covered before, since the configuration of the system was not available.
- Configuration errors, i.e., errors caused by erroneous or prohibited configuration.

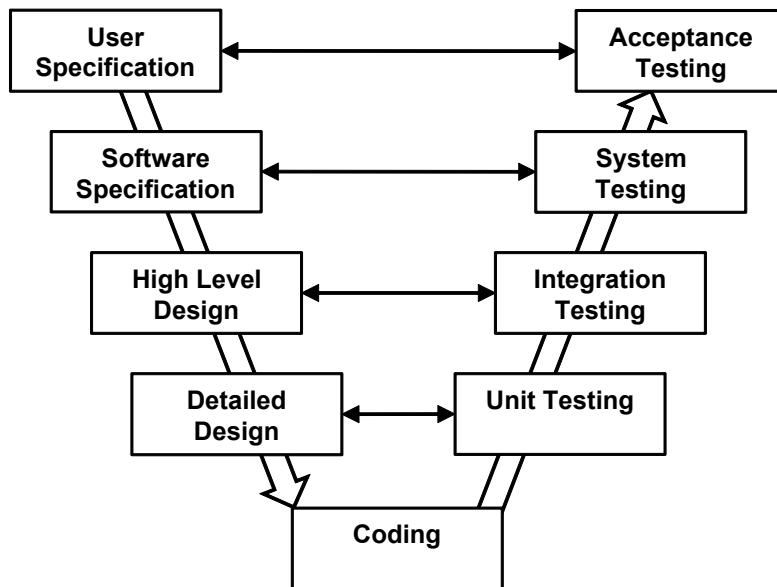


Figure 6.1: V-Model [111]

Additionally, performance aspects of TP parallel routing under a predefined configuration variant are explored and documented with the help of combinatorial testing.

The next subsections give a detailed description on how to implement steps of the combinatorial test process in the case of verifying TP parallel routing.

6.1 Building an IPM

As mentioned previously, IPM is a possible representation of the input space of the test object. It is required for any combinatorial test process. Generally, IPM is constructed from available system specifications. It is obvious that informal and semi-formal specifications lead mostly to different representations due to different interpretation possibilities [112]. Even formal specification can result in more than one possible IPM based on the experiences and the creativity of the test engineer.

An IPM consists of parameters with associated values used to practice properties of the test object. In the original approach, parameters of the test object are mapped one-to-one onto parameters of the IPM. Subsequently, values of IPM parameters are reduced by utilizing methods such as equivalence partitioning and boundary analysis [75]. However, in many new and innovative approaches, parameters are collected together into groups to represent functionalities of

the test object instead of the parameters themselves. In this way, the number of test cases shall be reduced and test cases shall be more effective.

Generally, grouping of input parameters in an efficient way requires more details about the test object and depends to a high level on the experiences of the test engineer, i.e., grouping of parameters by different persons will result in different IPMs.

To build an IPM for TP parallel routing, several actions must be performed. These are explained below.

6.1.1 Creating an Interaction Model

In order to define and create an Interaction Model of the IPM that supports the verification of TP parallel routing of an AUTOSAR gateway system, the approach depicted in fig. 6.2 is suggested.

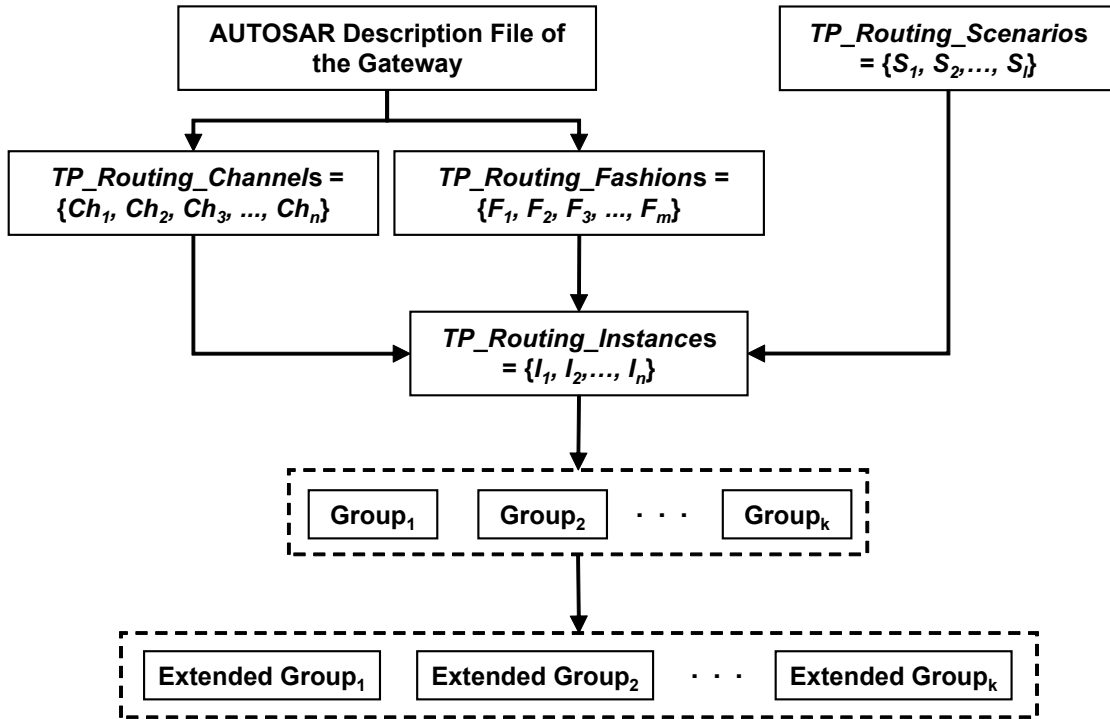


Figure 6.2: IPM for TP parallel Routing of AUTOSAR Gateway

According to the AUTOSAR methodology [14], an AUTOSAR description file must be available for any system that utilizes the standard. The AUTOSAR file uses a format similar to xml, called arxml, to describe the system on a specific abstraction level. It serves in most cases as a specification file. ARXML based files are semi-formal specifications which contain information about the system design. Input as well as output parameters of an AUTOSAR system are part of its AUTOSAR description file. Hence, this file is used as an input for the construction of IPM.

As depicted in fig. 6.2, several constructs are required to build up the desired Interaction Model (see fig. 5.1 for more details about the Interaction Model). Some of these constructs are formed automatically with the help of scripts (programs), while others have to be formed manually. To create the required constructs, several procedures are deployed as follows:

1. **Determine *TP_Routing_Scenarios***

Commonly, *TP_Routing_Scenarios* are determined through discussions with the persons who practice TP routing functionalities. Different advantages are gained from this procedure:

- Real use case scenarios are mostly not described and cannot be recognized or built automatically from the AUTOSAR description file of the gateway.
- Some use case scenarios are theoretically conceivable, however, they are not practiced.
- Conflicts between scenarios are also determined. These conflicts describe which scenarios are/aren't practiced in parallel.
- Future real use case scenarios can be discussed and considered for the test.

Two records are the output of this procedure. The first record, called *TP_Routing_Scenarios*, is used for the build process of the Interaction Model and has the structure as in (6.1):

$$[TP_Routing_Scenarios] = \begin{matrix} \{ \\ S_1, \\ S_2, \\ \cdot \\ \cdot \\ \cdot \\ S_1 \\ \} \end{matrix} \quad (6.1)$$

where S_1, S_2, \dots, S_1 are the determined *TP_Routing_Scenarios*.

The second record, called *TP_Routing_Scenarios_Conflicts*, includes conflicts among scenarios and is used later for the conflict handling process. This record has the struc-

ture as in (6.2):

$$\begin{aligned}
 [TP_Routing_Scenarios_Conflicts] = & \\
 & \{ \\
 & \quad TP_Scenario_Conflict_1 \\
 & \quad TP_Scenario_Conflict_2 \\
 & \quad \cdot \\
 & \quad \cdot \\
 & \quad \cdot \\
 & \quad TP_Scenario_Conflict_c \\
 & \} \tag{6.2}
 \end{aligned}$$

This procedure is performed manually. However, it has to be done only once. After that, gathered information can be utilized for every new release of the system or for similar systems.

2. Abstract *TP_Routing_Channels* and Build *TP_Routing_Fashions*

In this procedure, *TP_Routing_Channels* with their associated parameters are abstracted from AUTOSAR description file of the gateway. In parallel, parameters of *TP_Routing_Channels* are analyzed in order to build patterns of available *TP_Routing_Fashions*, i.e., for all *TP_Routing_Channels* that have the same number and type of parameters, a unique *TP_Routing_Fashion* will be defined. The outputs of this step are records which can be saved in a text file. These records are either from the type *TP_Routing_Channel* (see (3.10)) or *TP_Routing_Fashion* (see (3.1)).

With the help of a script, this procedure can be completely automated, since all required information is part of the AUTOSAR description file of the gateway. A script can parse the AUTOSAR file to abstract all available *TP_Routing_Channels* with their related parameters and analyze parameters to build *TP_Routing_Fashions*.

3. Build *TP_Routing_Instances*

As described previously, a *TP_Routing_Instance* is a relationship between a specific *TP_Routing_Channel* and a possible *TP_Routing_Scenario*. This relationship can be automatically resolved with the help of *TP_Routing_Fashions*, i.e., in order to practice a particular *TP_Routing_Scenario*, a *TP_Routing_Channel* from a specific *TP_Routing_Fashion* is implemented. In this procedure, records of *TP_Routing_Instances* are formed by assigning the right *TP_Routing_Scenarios* to individual *TP_Routing_Channels*. At this phase of the build process, the IPM can be seen as one-to-one assignment of the input/output parameter space.

4. Group *TP_Routing_Instances*

Grouping is a procedure in which *TP_Routing_Instances* that stimulate similar behavior in the gateway are clustered into groups. Two *TP_Routing_Instances* are said to

be similar if and only if they have the same number and values for all related parameters excluding the *Request_Identifier* and *Response_Identifier*. Creating groups of similar *TP_Routing_Instances* helps to reduce the number of test cases once combinations have to be built. Following example explains the reduction achieved with the help of groups:

Assume that the gateway test object has 4 *TP_Routing_Instances* A, B, C and D (see fig. 6.3) and is configured to support 2 *TP_Routing_Instances* in parallel. The number of possible combinations of 2 instances out of 4 would be 6 (the order has no effect). If *TP_Routing_Instances* A, B and C, D are respectively similar to each other, then groups can be constructed based on similarity criteria such that Group₁ consists of instances A and B, whereas Group₂ consists of instances C and D. After grouping, the number of combinations could be reduced from 6 to 3, since all other possible combinations would resemble a similar behavior, i.e., combinations of instances (A, C), (A, D), (B, C) and (B, D) are all similar and can be replaced by only one substitute (A, C). Hence, the combinations (A,B), (C,D) and (A,C) are sufficient to verify the system in this case.

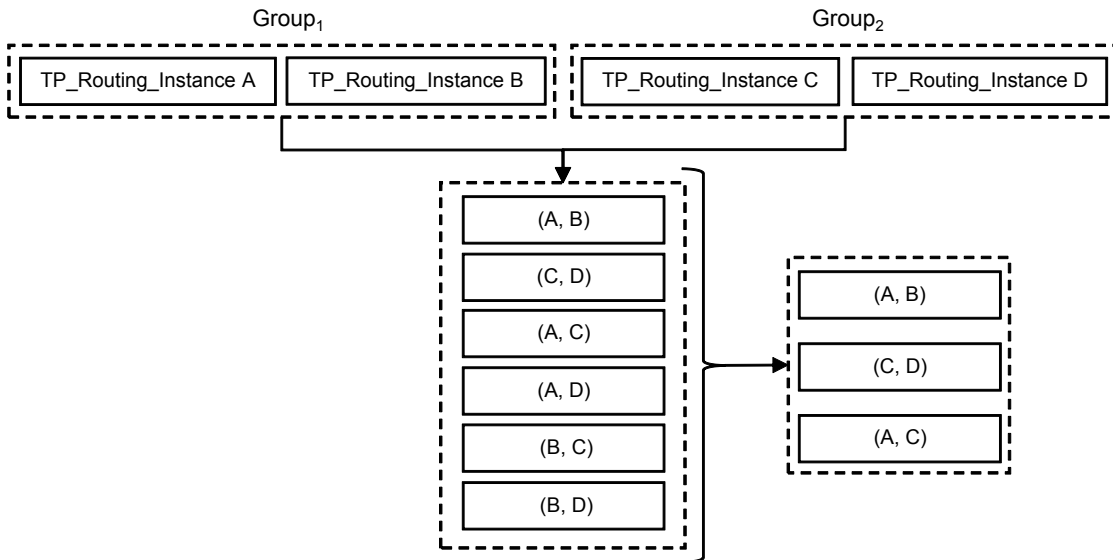


Figure 6.3: Advantages of Building Similar Groups for TP Parallel Routing

With the help of a script, this procedure can be completely automated. As a result, records of *TP_Routing_Instances* are collected into groups. The number of resulting groups depends on the grade of diversity in parameters of *TP_Routing_Instances*.

5. Extend Groups of *TP_Routing_Instances*

In this procedure, two additional parameters are assigned to individual groups. The first parameter (*Similarity Number*) is an integer variable used to recognize similar groups. Two groups are said to be similar if and only if their *TP_Routing_Instances* have the same number and values for all related parameters excluding the following parameters:

Source_Network, *Destination_Network*, *Request_Identifier* and *Response_Identifier*. Nevertheless, the source and destination networks of *TP_Routing_Instances* of similar groups must have the same characteristics, e.g., the same type and bandwidth of bus communication protocol. In this case, similar groups will be assigned the same value for *Similarity Number* parameter.

The value of the second parameter (*Stress Factor*) is calculated dynamically during test case execution based on aspects such as processing time, memory usage and channel bandwidth.

Introducing the *Similarity Number* and *Stress Factor* parameters helps to reduce the number of combinations to be tested as described later in the approach.

Since *TP_Routing_Instances* represent relationships, in which input and output parameters of the system are available, the Interaction Model is considered to have the form of an input-output parameter model.

6.1.2 Interaction Strength

Since no knowledge is available in advance about the effect of individual *TP_Routing_Instances* and their combinations on the behavior of the AUTOSAR gateway, the Uniform Strength Interaction is considered as interaction strength at the beginning of the test run. However, as test cases are constructed and run recursively in the proposed approach (see 6.6.1), the parameter *Stress Factor* of included *TP_Routing_Instances* is calculated. This parameter helps to determine different interaction strengths of *TP_Routing_Instances* and hence introduce the Variable Strength Interaction.

6.1.3 Creating a Conflict Model

During the construction process of IPM, conflicts must be resolved in order to guarantee a valid generation of test cases.

As explained previously, a conflict in IPM refers to invalid combination of parameters or parameter values. Hence, conflicts can lead to the generation of wrong test cases and must be handled.

In order to handle conflicts of an IPM, conflicts must be determined. In the case of TP parallel routing of an AUTOSAR gateway, three types of conflicts can be resolved:

1. **Type_A-Conflicts:** Conflicts between *TP_Routing_Scenarios*. These are resolved with the help of constraints describing *TP_Routing_Scenarios* not practiced in parallel (see the generated record [*TP_Routing_Scenarios_Conflicts*]).
2. **Type_B-Conflicts:** Conflicts between *TP_Routing_Channels*. These are resolved with the help of rules to recognize *TP_Routing_Channels* not allowed to be practiced in parallel. The usage of these rules depends on the utilized bus communication protocols of routing. Rules are saved as conditions for bus communication protocols.

3. **TypeC-Conflicts:** Conflicts because of configuration parameters. These are constraints describing known or desired capabilities of the gateway. These conflicts are also saved as conditions.

To explain these types of conflicts, assume an example of a gateway that has six *TP_Routing_Instances* used to route TP data between its connected ECUs as follows:

$TP_Routing_Instance_1 = (0x450, 0x5d9, 1, 2, 8, 8, 32, 8, 0, 10, Flashing)$
 $TP_Routing_Instance_2 = (0x411, 0x4c1, 2, 1, 8, 8, 8, 32, 10, 10, 11, 11, Uploading)$
 $TP_Routing_Instance_3 = (0x456, 0x5d5, 1, 2, 8, 8, 32, 8, 0, 10, Flashing)$
 $TP_Routing_Instance_4 = (0x411, 0x4c1, 2, 1, 8, 8, 8, 32, 10, 10, 12, 12, Uploading)$
 $TP_Routing_Instance_5 = (0x7f1, 1, 2, 8, 8, OBD)$
 $TP_Routing_Instance_6 = (0x7f1, 1, 3, 8, 8, OBD)$

Assume also that the gateway supports two *TP_Routing_Instances* in parallel. In this case, following examples can explain the three types of conflicts:

- **TypeA-Conflicts:** *TP_Routing_Scenario Flashing* and *TP_Routing_Scenario Uploading* are non-combinable. That is, a combination of *TP_Routing_Instance₁* and *TP_Routing_Instance₂* is for example an invalid combination.
- **TypeB-Conflicts:** *TP_Routing_Channel₂* and *TP_Routing_Channel₄* are non-combinable due to a rule defining that *TP_Routing_Channels* which have the same value for parameter *Request_Identifier* are non-combinable.
- **TypeC-Conflicts:** Due to the condition defining that only a maximum of two *TP_Routing_Instances* is supported, all combinations of more than two *TP_Routing_Instances* are invalid.

6.2 Definition of a Coverage Level

The definition of an appropriate coverage level is essential to combinatorial testing, since the coverage level affects the complexity and the thoroughness of the resulting test suit [90]. Many combinatorial test strategies support merely a certain coverage level. Therefore, the decision on which combinatorial strategy to apply may have an impact on the definition of the coverage level and vice versa.

In the proposed methodology for applying combinatorial testing to verify TP parallel routing of an AUTOSAR gateway system and measuring its performance, the coverage level is defined with respect to combinations of *TP_Routing_Instances* from constructed extended groups. For example, *I-wise* coverage requires that at least one possible combination with maximum supported *TP_Routing_Instances* from every input group is included at least once in a test case. Such combinations from groups are sufficient enough to satisfy *I-wise* coverage criterion, since all *TP_Routing_Instances* of a group stimulate a similar behavior of the gateway.

On the other side, *N-wise* coverage criterion (exhaustive testing) requires that all possible combinations with maximum supported *TP_Routing_Instances* from *N* input groups are included in some test cases.

The proposed methodology in this thesis supports *N-wise* coverage, where the design of coverage level can be described as follows:

For an arbitrary number of input groups, each with an arbitrary number of *TP_Routing_Instances*, the algorithm generates a *Combination Table* to satisfy *N-wise* interactions between the groups. The maximum number of resulting combinations is calculated as in (6.3):

$$X = \frac{(n + y - 1)!}{y!(n - 1)!} \quad (6.3)$$

Where *n* is the number of input groups and *y* is the maximum supported number of *TP_Routing_Instances* in parallel. Table 6.1 represents a *Combination Table* example of the defined *N-wise* coverage level. This example assumes an input of 4 groups (*G*₁,*G*₂,*G*₃,*G*₄) and a maximum of 3 *TP_Routing_Instances* supported to be practiced in parallel. The maximum number of resulting combinations (rows) in this example can be calculated according to (6.3) and is equal to 20 (20 rows).

Table 6.1: Example of a *Combination Table*

G₁	G₂	G₃	G₄
3	0	0	0
2	1	0	0
2	0	1	0
2	0	0	1
1	2	0	0
1	1	1	0
1	1	0	1
1	0	2	0
1	0	1	1
1	0	0	2
0	3	0	0
0	2	1	0
0	2	0	1
0	1	2	0
0	1	1	1
0	1	0	2
0	0	3	0
0	0	2	1
0	0	1	2
0	0	0	3

Rows of the table 6.1 represent combinations, while the numbers in columns represent the

count of *TP_Routing_Instances* that should be selected from corresponding groups to generate test cases.

Consider for example the second row of table 6.1. This row represents a combination in which two *TP_Routing_Instances* from group G_1 and one *TP_Routing_Instance* from group G_2 shall be selected to generate a test case covering that combination.

Since all *TP_Routing_Instances* of a group are similar to each other, including any of them will be appropriate to cover the combinations represented in rows, e.g., selecting any two *TP_Routing_Instances* from group G_2 for the combination in the second row will lead to the same behavior.

While generating test cases to cover combinations represented in rows, following additional aspects shall be considered:

- Each *TP_Routing_Instance* must be included at least once in some combinations. This is not required to satisfy the coverage level. However, it is a good practice to cover all available *TP_Routing_Instances*. That is, for the second row in table 6.1, 2 *TP_Routing_Instances* different than the 3 *TP_Routing_Instances* for the first row shall be selected from group G_1 .
- The sum of numbers in a column must be greater than or equal to the number of elements in the corresponding group in order to guarantee that each *TP_Routing_Instance* has been included at least once. If this is not the case, the remaining *TP_Routing_Instances* from that group shall be included and tested individually. These additional test cases are also not related to the coverage level. However, they are important to verify the functionality.
- The number in each column must be lesser than or equal to the total number of combinable *TP_Routing_Instances* in the corresponding group (see the next section for conflict handling). Otherwise, the combination in the related row is invalid and must be omitted.

To compare the reduction achieved in this example, let us assume that each of the 4 groups of the previous example has only 3 *TP_Routing_Instances*. The total number of input settings in this case would be equal to 12 and the resulting combinations without building groups would be equal to 220 (as calculated in (6.4)). This corresponds to a reduction of 90% of input settings.

$$\frac{(12)!}{3!(12-3)!} = 220 \quad (6.4)$$

6.3 Conflict Handling Strategy

Two basic rules must be retained while conflicts of the IPM are handled. Firstly, the defined coverage level must be satisfied and secondly, the test suit must be reduced since some combinations shall be avoided. In [113], four different methods for handling conflicts in IPMs were investigated. The result of the study was that the *avoid* method is best suited with respect to the size of the test suite if it can be utilized. To handle conflicts with the *avoid* method; a procedure is integrated in the test case selection algorithm to prohibit the selection of conflicting combinations. Another method mentioned in the study is the *sub-models* method, in which conflicts

are removed by splitting the IPM into multiple smaller conflict-free IPMs used separately for test case selection and generation. In this thesis, a combination of these two conflict handling methods is utilized to handle the determined conflicts during the test of TP parallel routing.

6.3.1 Type_A-Conflict Handling

Since constructed Extended Groups possess *TP_Routing_Instances* similar to each other, *TP_Routing_Instances* of each Extended Group will have the same attached *TP_Routing_Scenario*. To handle conflicts between *TP_Routing_Scenarios*, the *sub-models* method is utilized. Thereby, the parameter *TP_Routing_Scenario* is used to split IPM into sub-IPMs which are Type_A-Conflict-free.

As depicted in fig. 6.4, Extended Groups along with the record [*TP_Routing_Scenarios_Conflicts*] are the input for the implementation of *sub-models* method. In order to create sub-IPMs which are Type_A-Conflict free, the *sub-models* method processes the record [*TP_Routing_Scenarios_Conflicts*] to collect Type_A-Conflict free Extended Groups and create Type_A-Conflict free sub-IPMs. This procedure is performed automatically and shall be completed before *Combination Tables* of the defined coverage level are generated.

Resulting sub-IPMs are then processed successively, where for each sub-IPM a *Combination Table* is generated and handled separately.

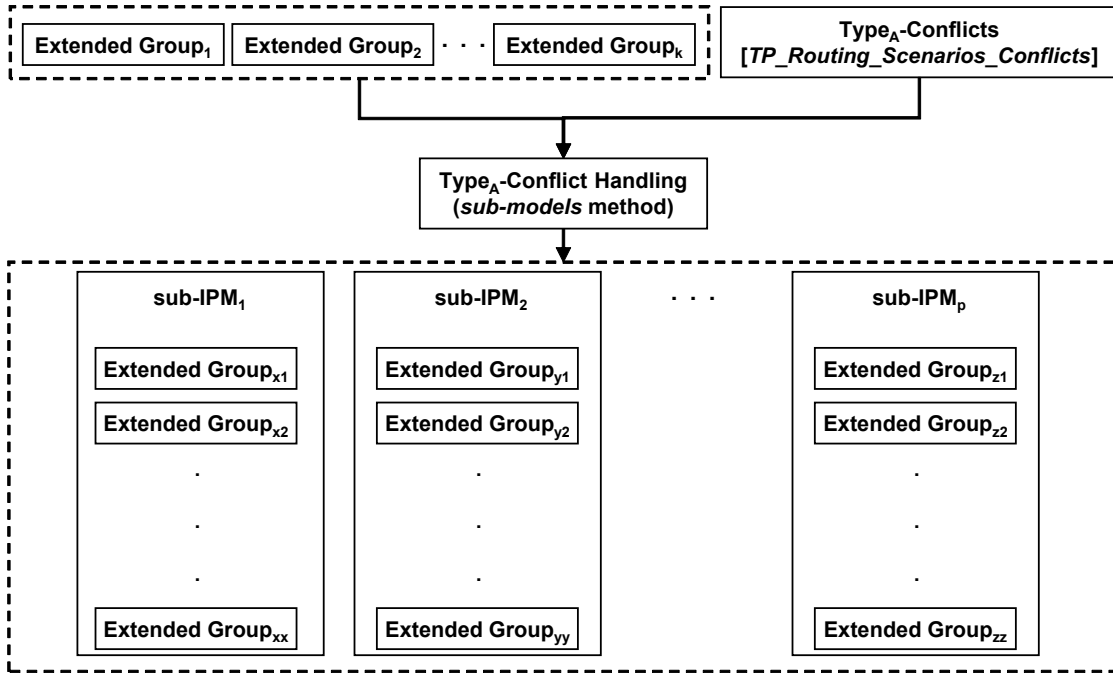


Figure 6.4: Type_A-Conflict Handling Approach

6.3.2 Type_B-Conflict Handling

As explained previously, Type_B-Conflicts describe impossible combinations of *TP_Routing_Channels* which are the main components to build *TP_Routing_Instances*. In order to handle this kind of conflicts, the *avoid* method is utilized.

After *Combination Tables* are built for Type_A-Conflict free sub-IPMs from the previous step, the *avoid* method is applied to each *Combination Table* separately and in a recursive manner.

To automate the implementation of the *avoid* method, three reserved variables are attached to individual *TP_Routing_Instances* of input groups. These are *Conflict ID*, *Include Times* and *Token*.

The *Conflict ID* variable identifies which *TP_Routing_Instances* are non-combinable due to Type_B-Conflicts. That is, *TP_Routing_Instances* which are conflicting due to the same Type_B-Conflict, will be assigned the same value for variable *Conflict ID* and are non-combinable with each other. However, Type_B-Conflict free *TP_Routing_Instances* will be assigned the special value “0” for *Conflict ID* and can be always combined.

The *Include Times* variable holds the number of times a *TP_Routing_Instance* has been selected to build combinations. As mentioned previously, *TP_Routing_Instances* of an Extended Group are similar to each other, and the selection of any of them is sufficient to satisfy a defined combination from the *Combination Table*. Therefore, this variable is utilized to choose *TP_Routing_Instances* from an Extended Group that have not been yet included or less included than other instances to guarantee a fair coverage of available *TP_Routing_Instances*.

Every *TP_Routing_Instance* is assigned an *Include Times* value “0” at the initial phase of the selection process. This number gets incremented whenever that particular *TP_Routing_Instance* becomes part of a generated combination.

Since the selection of *TP_Routing_Instance* to build combinations is performed in a recursive manner as explained later, the value of the third variable *Token* is used to decide whether a *TP_Routing_Instance* is considered for the next selection or not. That is, for every loop of the selection and generation process, the so-called *Token* is rotated to another *TP_Routing_Instance* to avoid conflicts. The *Token* variable can have one of the following values:

- **0:** denotes that related *TP_Routing_Instance* has no Type_B-conflict and can be included in any combination and in every selection loop.
- **1:** denotes that related *TP_Routing_Instance* has Type_B-conflict and can be considered only for the next selection loop.
- **2:** denotes that related *TP_Routing_Instance* has Type_B-conflict and is not considered for the next selection loop.

The algorithm for handling Type_B-Conflicts with the *avoid* method is depicted in fig. 6.5. The algorithm describes the procedure applied to one Type_A-Conflict free sub-IPM. The procedure begins with the assignment of values to variables *Conflict ID* of *TP_Routing_Instances*. This is achieved based on defined rules for Type_B-Conflicts and follows the concept described previously. In the next step, initial values for variables *Token* and *Include Times* are assigned. Hereby, all *Include Time* variables will get the initial value 0, whereas, the initial value for variables *Token* depends on the *Conflict ID* values.

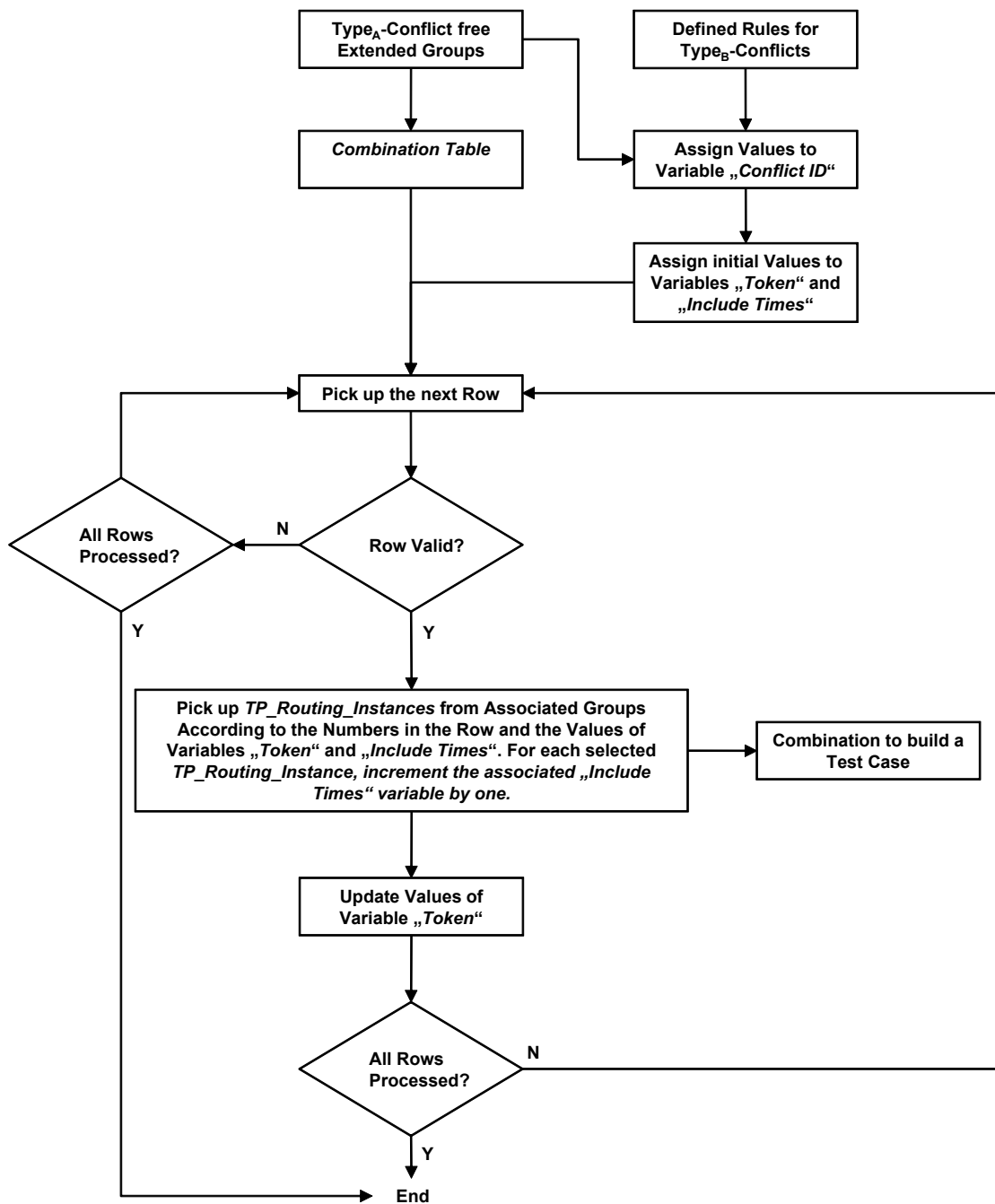


Figure 6.5: Type_B-Conflict Handling Approach

Since *TP_Routing_Instances* that have the same *Type_B-Conflict* are assigned the same value for variable *Conflict ID*, only one of them will get the initial value 1 for variable *Token* for the next selection and generation loop. Other *TP_Routing_Instances* with the same *Conflict ID* value will get the initial value 2. *TP_Routing_Instances* which are *Type_B-Conflict* free will get the initial value 0 that remains constant during the whole procedure.

After generating the *Combination Table* for Extended Groups of a sub-IPM, the algorithm processes its rows successively. It picks up one row and check its validity before selecting *TP_Routing_Instances* from groups. For the case that one row is invalid, the whole row (combination) is avoided. Validity check depends on the numbers in processed row and combinable *TP_Routing_Instances* of related groups. If at least one number of the row is greater than combinable *TP_Routing_Instances* of related group, the row is considered as invalid. To determine the count of combinable *TP_Routing_Instances*, the variable *Token* is used. The method calculates the number of combinable *TP_Routing_Instances* based on the values 0 and 1 for the *Token* variable.

Avoiding invalid rows of *Combination Table* does not affect the coverage level, since the number of *TP_Routing_Instances* that shall be selected from one group of invalid row is impossible, and the interaction of the invalid row shall be implicitly covered in other rows. To explain this, consider again the *Combination Table* example in 6.1. Assume that only one *TP_Routing_Instance* of group *G₂* is combinable at a time, i.e., rows 5, 11, 12 and 13 are invalid. Row 5 covers the interaction between 1 *TP_Routing_Instance* from group *G₁* and 2 *TP_Routing_Instances* from group *G₂*. As only 1 *TP_Routing_Instance* from group *G₂* is combinable at a time, the combination of row 5 should be replaced by 1,1,0,0. This replacement is implicitly part of rows 2,6 and 7 and considering it again is redundant. Therefore, the row 5 shall be avoided. The same is applied for rows 11, 12 and 13.

For the case that a row is valid, *TP_Routing_Instances* from input Extended Groups are picked up according to the numbers in row as well as the values of variables *Token* and *Include Times*. Only *TP_Routing_Instances* with the values 0 and 1 for variable *Token* are allowed to be selected to build a combination. Additionally, smaller values for the variable *Include Times* have priority over others. In the case that a decision has to be taken between *TP_Routing_Instances* which have the same values for variables *Include Times* and *Conflict ID*, the first one can be chosen. This will help to include all *TP_Routing_Instances* with the same repetition.

Once *TP_Routing_Instances* have been selected for one row, these will be used to construct a test case.

Updating the variable *Token* in the next step is performed based on the values of *Conflict ID* and *Include Times*. The idea is to give the value 1 to the *Token* variable of the next *TP_Routing_Instance* with the same *Conflict ID* and minimum *Include Times*. The procedure continues until processing all rows of *Combination Table* based on the same concept.

6.3.3 *Type_C-Conflict Handling*

As explained previously, *Type_C-Conflicts* are constraints that determine the maximum supported numbers of parallel *TP_Routing_Instances* of the deployed TP protocols. These constraints are handled during the construction process of *Combination Tables*. That is, the sum of numbers in rows of a *Combination Tables* must not exceed the maximum supported numbers of parallel

TP_Routing_Instances of TP protocols related to Extended Groups of columns. Since combinations and corresponding test cases are generated recursively from the *Combination Table*, these constraints can be corrected depending on the results of run test cases. The correction helps to determine the TP routing performance of the gateway, e.g., reducing the maximum number of supported parallel *TP_Routing_Instances* for CAN TP if a performance issue raised during TP routing. If these constraints have to be corrected during test run, the *Combination Table* must be respectively modified. In this case, invalid rows must be avoided or corrected for the next loops of the selection and generation process.

6.4 Computational Implementation

Since constructed sub-IPM are processed successively and independently from each other in the proposed approach, parallel computing is well-suited. However, the limited resources were deciding for the deployment of sequential computing in this thesis.

6.5 Test Suite Reduction

To achieve the goal of reducing the size of the test suite and still have the same desired coverage level, two additional constructs can be defined for individual Type_A-Conflict free sub-IPMs. The first construct is Single Network Relationship (SNR) and the second is Multiple Network Relationship (MNR).

Basically, several SNRs and merely one MNR are formed for one Type_A-Conflict free sub-IPM. These are used for the test process which is consequently carried out in two separate steps for individual sub-IPMs as follows:

1. Testing TP parallel routing for SNRs
2. Testing TP parallel routing for MNR

SNRs are built with the help of a similarity criterion. An individual SNR comprises Extended Groups of *TP_Routing_Instances* responsible for routing TP data between two specific networks of the gateway. That is, Extended Groups of a sub-IPM that have the same values for parameters *Source_Network* and *Destination_Network* shall form an individual SNR.

After SNRs are formed, the first test step is conducted and executed as described in details in the next section. While TP parallel routing for SNRs is carried out, the test results are analyzed in order to assign a suitable value for parameter *Stress Factor*. The *Stress Factor* parameter helps consequently to form MNR used in the second test step.

A MNR is built with the help of gained information from test cases executed for individual SNRs. The MNR of a sub-IPM is a collection of Extended Groups with one selected Extended Group from each SNR. The selection process is based on the values of parameter *Stress Factor*. From each SNR, the Extended Group with the best value for parameter *Stress Factor* is chosen to form the unique MNR of the processed sub-IPM.

Defining the two constructs SNR and MNR has two advantages. The first advantage is related to reducing the number of rows in the *Combination Table* and can be explained with the following example:

Assume that we have a Type_A-Conflict free sub-IPM which consists of 10 individual Extended Groups. Assume that *TP_Routing_Instances* of the 10 Extended Groups are responsible for routing TP data between CAN network. The maximum supported number of parallel *TP_Routing_Instances* on CAN is equal to 3. Let us assume that each Extended Group has at least 3 *TP_Routing_Instances* available to build combinations. According to the equation (6.3), the number of rows in the *Combination Table* for the 10 Extended Groups would be in this case 220.

Assume now that we can build 3 SNRs as follows: the first SNR has 3 Extended Groups, the second SNR has 3 Extended Groups and third SNR has 4 Extended Groups. For these SNRs, the three *Combination Tables* would have respectively the following number of rows according to the equation (6.3): 10, 10 and 20. Since three SNRs are built, the unique MNR would have 3 selected Extended Groups, and the *Combination Table* of MNR would contain 10 rows. Altogether, 50 rows and therefore 50 combinations will be generated from all *Combination Tables*. In this simple example, the number of row was reduced from 220 to 50 by utilizing the two constructs SNR and MNR.

The second advantage is concerned with reducing the search space for reasons of errors, once they are discovered. Since the test is carried out separately for SNRs and a SNR includes *TP_Routing_Instances* between two specific networks, discovered errors during the test of a particular SNR would give hints that errors are possibly related to specific networks. Additionally, if similar errors are discovered during the test of SNRs with the same types of networks, this would then give hints that errors are possibly related to a specific type of networks and so on.

Testing SNRs and MNRs is described in details in the next section.

6.6 Test Case Generation, Execution and Evaluation

6.6.1 Testing TP parallel routing for SNRs

The procedure for testing TP parallel routing for SNRs is depicted in fig. 6.6. The procedure accepts Type_A-Conflict free sub-IPMs as input and processes them successively. For each sub-IPM, the *Selector ()* function extracts Extended Groups of the next SNR. With the help of the mechanism explained previously to satisfy *N-wise* coverage level, the *Combination Table* is built for groups of the current processed SNR. Subsequently, Type_B-Conflict handling is conducted to select recursively *TP_Routing_Instances* according to rows of the *Combination Table*. For each produced combination of *TP_Routing_Instances*, some additional information are added to generate a runnable test case. In the next step, the generated test case for processed combination is executed and the test results are analyzed. Diverse aspects are analyzed from the results of executing test cases, such as the time required to route TP data using *TP_Routing_Instances* from specific Extended groups and the behavior of TP data routing during the whole routing process. Based on this analysis, values for parameter *Stress Factor* of individual Extended Groups are

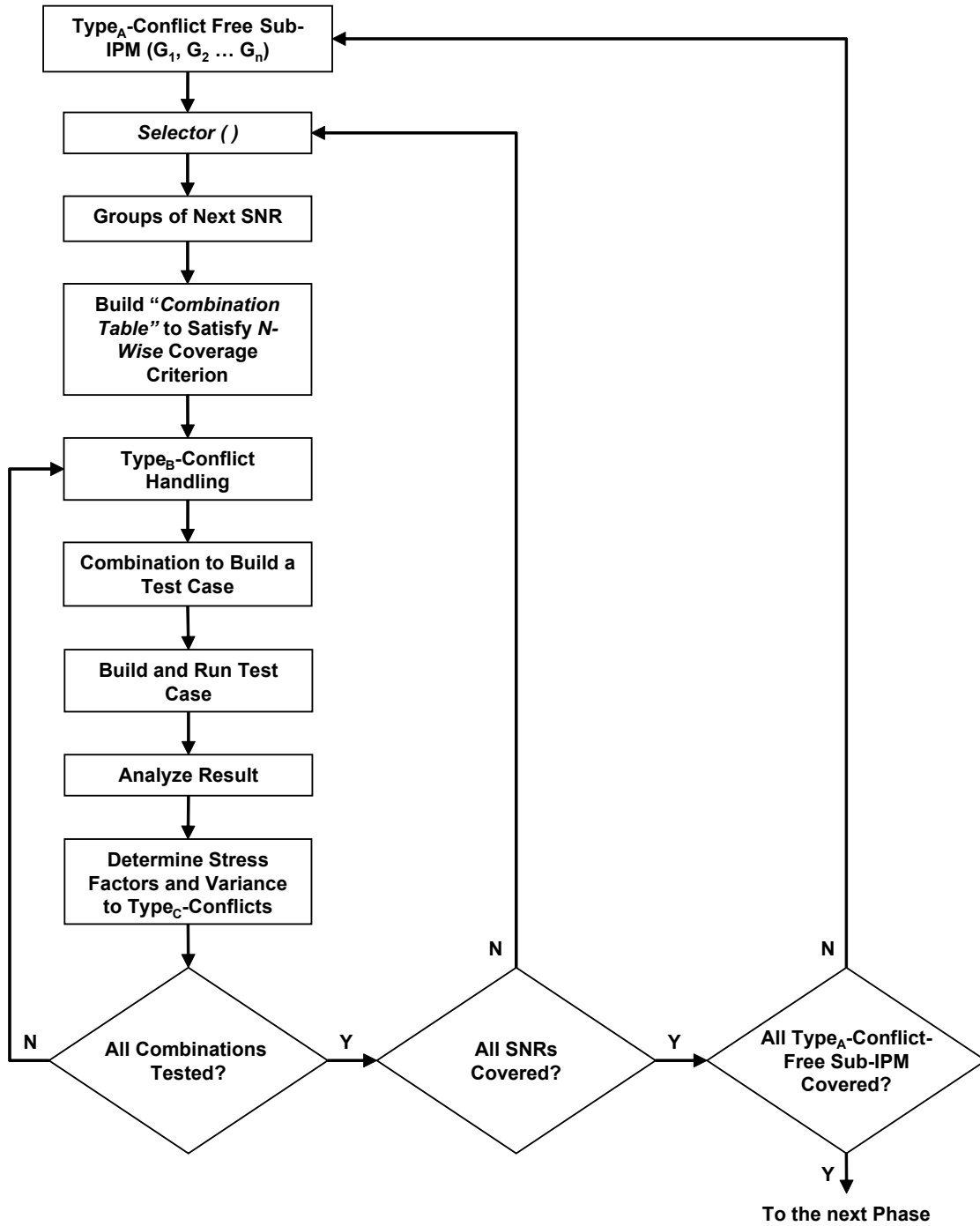


Figure 6.6: TP Parallel Routing for SNRs

assigned and variance to Type_C-Conflicts is determined to correct the constraints and measure the performance.

This procedure is repeated for all rows of the *Combination Table* until the *N-wise* coverage level is satisfied for each SNR. The same process is repeated until all Type_A-Conflict free sub-IPM are processed.

After the completion of this step, each Extended Group will be assigned a calculated *Stress Factor*, which will be used by the next test step.

6.6.2 Testing TP parallel routing for MNRs

The procedure for testing TP parallel routing for MNRs is depicted in fig. 6.7. In this procedure, Type_A-Conflict-free sub-IPMs are processed successively. As mentioned previously, an arbitrary number of SNRs is created for each sub-IPM. To build the MNR for a sub-IPM, a representative Extended Group from each of its SNRs is selected. The selection depends on calculated *Stress Factor* of Extended Groups. That is, the Extended Groups with the best values for parameter *Stress Factor* build the MNR of a sub-IPM and are the basis for testing in this step. Once Extended Groups of MNR are determined, the groups that have the same *Similarity Numbers* will be omitted in order to avoid similar combinations. Then, the same procedure as in the previous step is utilized to build combinations and execute test cases until *N-wise* coverage level is satisfied. The procedure stops once all Type_A-Conflict-free sub-IPMs are processed. During recursive testing of MNR, constraints of Type_C-Conflicts are also corrected if an error is observed. With the help of the correction, further combinations are reduced and the performance of the gateway can be determined.

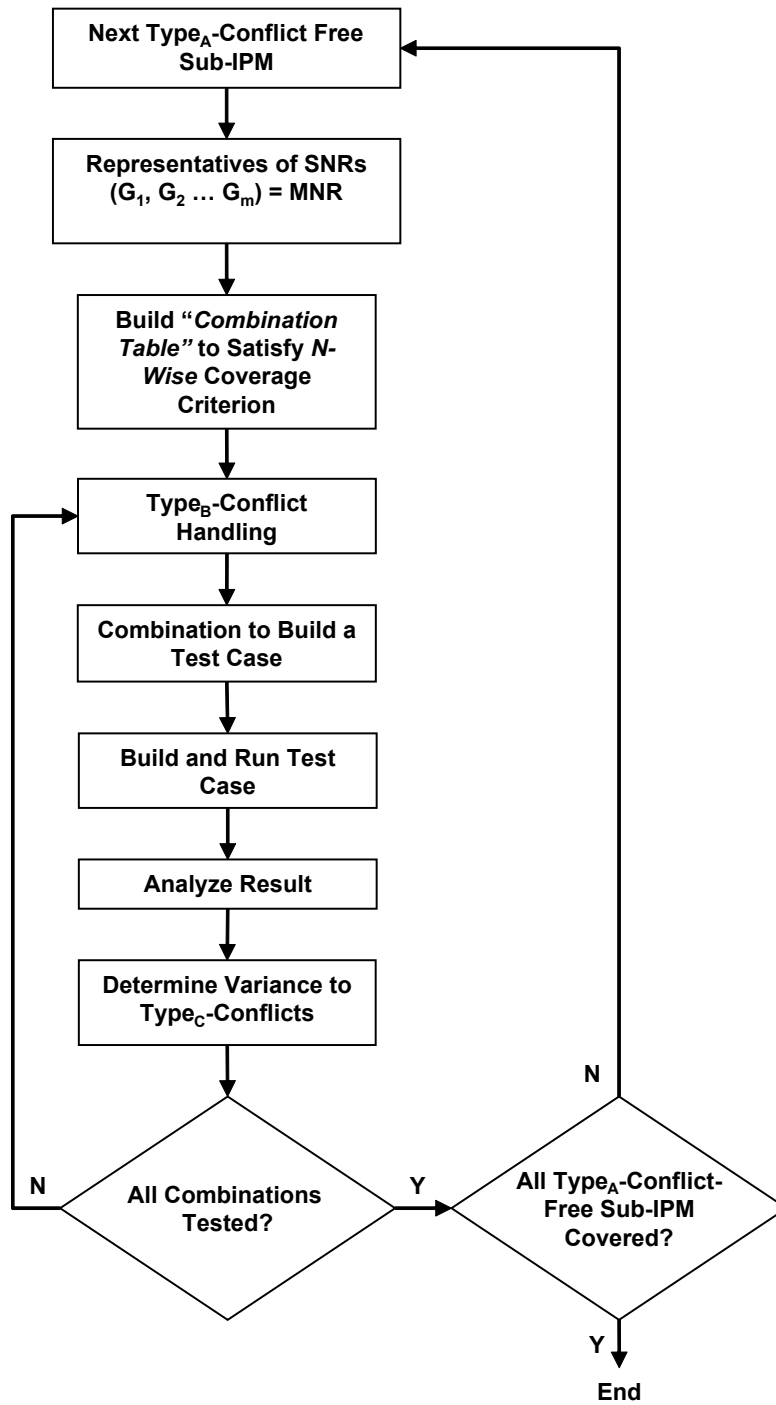


Figure 6.7: TP Parallel Routing for MNRs

Validation and Evaluation

In this chapter, the proposed combinatorial test approach is validated by means of two examples of real AUTOSAR gateway systems. Moreover, the results achieved by applying the test approach are evaluated to show its effectiveness to reduce the number of combinations to be tested and hence the test time.

7.1 Implementation of the Test System

In this section, the test system which implements the proposed combinatorial test approach is presented. Figure 7.1 depicts the design of the test system. As shown in the figure, the test system consists of multiple parts responsible for realizing the approach discussed previously as well as implementing a restbus simulation required to execute selected and generated test cases. Additionally, the test system design consists of a reporting part to document generated test cases and the information related to their execution.

The test system works automatically. It uses as input the description file of the gateway test object (AUTOSAR ECU Description File) and some manual definitions. The manual definitions are created during the design phase of the test system. Consequently, gateway test objects can be evaluated automatically without any further manual intervention. However, the AUTOSAR ECU Description File must be adapted each time a new gateway test object or a new version of the gateway test object has to be verified.

7.1.1 Manual Definitions

The automated test system requires onetime manual definition of the following records:

- [*TP_Routing_Scenarios*]: The record in which scenarios utilizing TP routing functionality of the gateway system are registered.
- [*TP_Routing_Fashions*]: The record in which the number and type of parameters required for each *TP_Routing_Scenario* are registered.

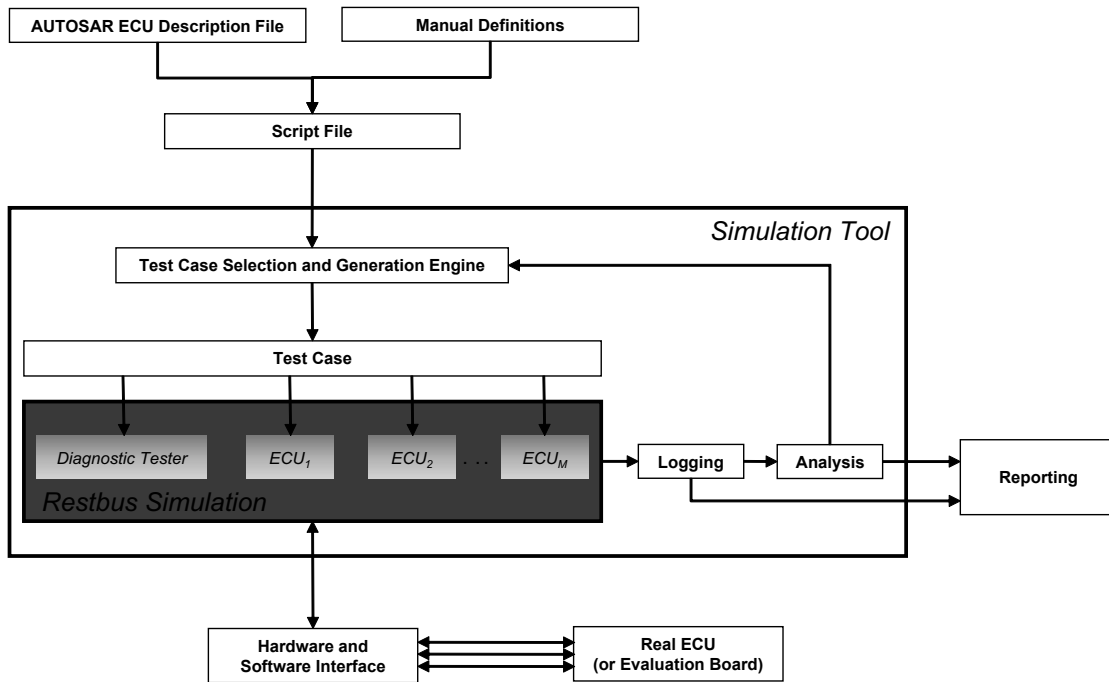


Figure 7.1: Test System Design

- **[TP_Routing_Scenarios_Conflicts]:** This record contains conditions or constraints of combinable *TP_Routing_Scenarios*.
- **[TP_Routing_Connections_Conflicts]:** The record in which rules for Type_B-Conflicts are specified. This record helps to recognize conflicting *TP_Routing_Channels* automatically.

7.1.2 Parameter Abstraction and IPM

According to the AUTOSAR standard, an AUTOSAR based gateway must have an AUTOSAR Description File that contains information about the gateway test object. Information regarding TP routing is part of this file.

As mentioned previously, an AUTOSAR Description File has a format similar to XML, called ARXML, which is human- and machine-readable. In order to abstract TP relevant parameters from the AUTOSAR Description File of the gateway test object, a script written in the scripting language “Perl” is used. The *Perl* script works with any AUTOSAR Description File uses the specification of the system template V3.6.0 R3.2 Rev 3 [37]. An AUTOSAR system template contains “*keywords*” that can be used to search and abstract required information related to TP routing on CAN, FlexRay and LIN bus systems. The *Perl* script uses among others the following *keywords* (written in cursive) of the AUTOSAR Description File to search and abstract TP information required to implement the proposed approach:

- *Keywords* of **PDU Router** Module

1. *PduR*: Keyword used to search for the container of the configuration parameters of PDU Router module.
2. *PduRGeneral*: Keyword used to search for the subcontainer of PduR container that contains general configuration parameters of PDU Router module, as for example the parameter *PDUR_MEMORY_SIZE* which defines the memory size reserved for PDU Router buffers and *PDUR_GATEWAY_OPERATION* which is used to enable and disable PDU Router gateway operation.
3. *PduRTpBufferTable*: Keyword used to search for the subcontainer of PduR container that contains the definition of all TP buffers. The configuration parameter *PDUR_MAX_TP_BUFFER_NUMBER* which defines the maximum number of TP buffers is included in this container.
4. *PduRTpBuffer*: Keyword used to search for the container of *PduRTpBufferTable* subcontainer that specifies a TP buffer. The configuration parameter *Length* which defines the length of the buffer is included in this container.
5. *PduRRoutingTable*: Keyword used to search for the subcontainer of *PduR* container that contains all routing paths of PDUs.
6. *PduRRoutingPath*: Keyword used to search for the container of *PduRRoutingTable* subcontainer that specifies the routing path of a PDU. Configuration parameters *SduLength* to define the length of PDU and *TpChunkSize* to define the chunk size are included in this container.
7. *PduRSrcPdu*: Keyword used to search for the container of *PduRRoutingPath* subcontainer that specifies the source of the PDU to be routed.
8. *PduRDestPdu*: Keyword used to search for the container of *PduRRoutingPath* subcontainer that specifies one destination of PDU to be routed.

For more *keyword* of PDU Router module see [114].

- **Keywords of FlexRay TP Module**

1. *FrTpConfiguration*: Keyword used to search for the container that contains general configuration parameters of the FlexRay TP module. Examples of configuration parameters included in this container are: *FRTP_CHAN_NUM* to define the number of concurrent channels supported, *FRTP_HAVE_ACKRT* to enable and disable the acknowledgment and retry mechanisms, *FRTP_HAVE_GRPSEG* to enable and disable segmentation of 1:n messages and *FRTP_HAVE_TC* to enable and disable transmit cancellation.
2. *FrTpChannelConfiguration*: Keyword used to search for the container that contains configuration parameters of one FlexRay TP channel. Examples of configuration parameters included in this container are: *FRTP_CHANNEL_ID* to define the ID of the channel, *FRTP_CON_NUM* to define the number of connections used in the channel, *FRTP_TIMEOUT_AS*, *FRTP_TIMEOUT_AR*, *FRTP_TIMEOUT_BS*, *FRTP_TIMEOUT_BR*, *FRTP_TIMEOUT_CS* and *FRTP_TIMEOUT_CR* that define

timing parameters of the channel, and *F RTP_ACKTYPE* to define the type of acknowledgment for the channel.

3. *FrTpConnectionConfiguration*: Keyword used to search for the container that contains configuration parameters of one FlexRay TP connection. Examples of configuration parameters included in this container are: *F RTP_CON_CHANNEL* that defines the ID of the channel to which the connection belongs to, *F RTP_SDUID* that defines the ID of the connection, *F RTP_LA* that defines the local address for the respective connection, and *F RTP_RA* that defines the remote address for the respective connection.

For more *keywords* of FlexRay TP module see [24].

- **Keywords of CAN TP Module**

1. *CanTp*: Keyword used to search for the container that contains configuration parameters of CAN TP module.
2. *CanTpGeneral*: Keyword used to search for the subcontainer of *CanTp* container that contains general configuration parameters of CAN TP module. Examples of configuration parameters included in this container are: *CanTpMainFunctionPeriod* that defines the cycle time of the main function of CAN TP module and *CanTpTc* that enable or disable transmit cancellation on CAN bus.
3. *CanTpRxNSdu*: Keyword used to search for subcontainers that contain configuration parameters of each configured CAN N-SDU received by CAN TP module. Examples of configuration parameters included in this container are: *CANTP_ADDRESSING_FORMAT* to define the addressing mode for the received N-SDU, *CANTP_BS* that defines the block size, *CANTP_NAR*, *CANTP_NBR* and *CANTP_NCR* that define timing parameters of N-SDUs, *CANTP_RX_CHANNEL* to define the link to the RX connection channel used to receive the N-PDU, *CANTP_WFTMAX* to define the maximum number of wait N-PDUs that can be consecutively transmitted by the receiver and *CANTP_STMIN* that defines the minimum separation time.
4. *CanTpTxNSdu*: Keyword used to search for the subcontainer that contains configuration parameters of each configured CAN N-SDU to be transmitted by the CAN TP module. Examples of configuration parameters included in this container are: *CANTP_ADDRESSING_MODE* that defines the addressing mode of the transmitted N-SDU, *CANTP_NAS*, *CANTP_NBS*, *CANTP_NCS*, *CANTP_NAS* that define timing parameters and *CANTP_DL* that defines the data length code of the transmitted N-SDU.

For more *keywords* of CAN TP module see [25].

- **Keywords of LIN TP Module**

1. *LinTp*: Keyword used to search for the descriptor of LIN TP module.

2. *LinTpGeneral*: Keyword used to search for the container that contains all LIN TP general parameters.
3. *LinTpRxNSdu*: Keyword used to search for containers of received N-SDUs on LIN channels. Configuration parameters included in each of these containers are for example, *LIN-TP_DL* which holds the data length of the received N-SDU, *LINTP_NSDU_ID* which holds the identifier of received N-SDU and *LINTP_NAD* which holds the NAD value for a specific LIN slave.
4. *LinTpTxNSdu*: Keyword used to search for containers of transmitted N-SDUs on LIN channels. Configuration parameters included in each of these containers are for example, *LINTP_NSDU_ID* which holds the identifier of transmitted N-SDU and *LINTP_NA-D* which holds the NAD value for a specific LIN slave.

For more *keywords* of LIN TP module see [115].

While parsing an AUTOSAR Description File, the consistency of configuration parameters related to TP routing is checked. The aim of consistency check is to figure out errors in the configuration. For example, a consistency check rule can examine if all required configuration parameters are available for *TP_Routing_Channels* of a specific *TP_Routing_Fashion*.

As the Script abstracts TP configuration parameters and builds *TP_Routing_Channels*, the manual definitions are used to assign suitable *TP_Routing_Scenarios* to *TP_Routing_Channels* in order to form *TP_Routing_Instances*. Once this process is completed, the Script compares parameters of *TP_Routing_Instances* to collect them into groups. Finally, the two additional parameters, *Similarity Number* and *Stress Factor*, are assigned to form Extended Groups.

After Extended Groups are formed, the Script deploys the *sub-models* method to handle Type_A-Conflicts. This is achieved by collecting the Extended Groups with the help of the record [*TP_Routing_Scenarios_Conflicts*]. The results of this step are sub-IPMs that are Type_A-Conflict free, where each sub-IPM consists of a number of Extended Groups.

Generally, it is usual that Extended Groups are part of more than one sub-IPM. This reflects the fact that some *TP_Routing_Scenarios* may be combined with multiple conflicting *TP_Routing_Scenarios*. To explain this, assume a simple example of three different *TP_Routing_Scenarios* as follows: S_1 is assigned to an Extended Group G_1 , S_2 is assigned to an Extended Group G_2 and S_3 is assigned to an Extended Group G_3 . Let us assume that S_1 and S_2 are conflicting. In this case, handling Type_A-Conflicts will result in the following sub-IPMs:

1. sub-IPM₁ with the Extended Groups (G_1 , G_3)
2. sub-IPM₂ with the Extended Groups (G_2 , G_3)

Here, the Extended Group (G_3) is part of two sub-IPMs.

The approach to handle Type_B-Conflicts is implemented as part of the test case selection process in the Test Case Selection and Generation Engine part of the test system as explained in the next subsection.

7.1.3 Test Case selection and Generation Engine

This part of the test system is responsible for the generation of *Combination Tables* for the different sub-IPMs as well as the selection of *TP_Routing_Instances* according to rows of the *Combination Tables*. It is also responsible for the creation of test cases as described in the proposed approach.

The test case selection and generation engine accepts Type_A-Conflict free sub-IPMs constructed in the previous parts as input and processes them successively. The engine deploys the following functions to implement its task:

- *selector(int Groups_References[][])*: The input of this function are groups of a Type_A-Conflict free sub-IPM which are referenced in the parameter array *Groups_References*. The function is used to create SNRs for its input groups.
- *initialize(int Groups_References[][])*: This function assigns initial values to parameters *Include Times*, *Conflict ID* and *Token* of *TP_Routing_Instances* of input groups that are referenced in the parameter array *Groups_References*. The initial values for parameter *Include Times* is 0 for all *TP_Routing_Instances*. However, the parameter *Token* will be initialized with one of the values 0,1 or 2 depending on the values of parameters *Conflict ID* and *Include Times*.
- *build_combination_table(int Groups_References[][])*: This function is used to build a *Combination Table* that covers *N-Wise* interactions between input groups. The function uses the parameter *Groups_References* to access its input groups.
- *next_row(int Combination_Table[][])*: This function moves the selection and generation process to the next row of the processed *Combination_Table*.
- *validate_row(int row[])*: This function checks the possibility to build a combination for a given row. The validity check is based on the availability of conflict-free *TP_Routing_Instances* in the related groups as specified in the row of the processed *Combination_Table*.
- *select_elements_of_row(int row[])*: This function picks up *TP_Routing_Instances* as specified in parameter *row* to build a combination. Values of parameters *Token* and *Include Times* are used as a selection criterion. Once a *TP_Routing_Instance* is included in a test case, its associated *Include Times* parameter will be incremented by one.
- *update_token()*: This function uses the current values of *Include Times* parameter along with the values of *Conflict ID* parameter to assign suitable values to *Token* parameter as described in the approach.
- *generate_test_case(int References[])*: This function supplements the selected *TP_Routing_Instances* with additional information necessary to generate an executable test case.
- *construct_MNR(int Groups_References[][])*: This function isolates representatives of SNRs of a Type_A-conflict free sub-IPM based on values of parameter *Stress Factors*. Subsequently and with the help of parameter *Similarity Number*, MNR is formed.

These function are utilized to implement steps of the approach included in figures 6.5, 6.6 and 6.7.

7.1.4 Restbus Simulation

The main task of the restbus simulation is to run generated test cases. To achieve this task, the restbus simulation has the ability to communicate with the gateway test object on its connected networks as in real scenarios. The access to networks of the gateway is supported by a Hardware and Software Interface (see fig. 7.1). Over this Interface, the restbus simulation can trigger TP routing by sending frames on the right networks and responding to outputs as depicted for example in fig. 3.4.

While executing a test case, the restbus simulation simulates the behavior of the gateway's environment according to parameters of a test case. That is, a test case provides the restbus simulation with parameters required to simulate the environment and validate the reactions.

The restbus simulation implemented in this thesis supports up to 4 CAN networks and one FlexRay network. The maximum number of supported parallel *TP_Routing_Instances* is 40. These two restrictions can be easily extended to support higher capabilities of the gateway test object. However, an extension is not required if the gateway test object supports the same set or a subset of these features.

Time in the Restbus Simulation

Since TP routing is a real-time functionality which utilizes timing parameters to control and monitor the communication, time shall be handled very carefully in the restbus simulation.

Mainly, two timing aspects shall be considered. The first aspect is concerned with monitoring and validating the reaction time of the gateway test object, while the second aspect focuses on the moment of time in which the restbus simulation shall react to outputs of the gateway test object. Additionally, a timeout for running of test cases shall be defined to prevent infinite loop of execution.

During the execution of a test case, timing aspects of the environment are handled by means of two different techniques. The first technique uses timers supported in most simulation tools, whereas the second technique is based on the calculation of differences between timestamps.

Due to the uncontrollable additional time required to start and stop timers in most simulation tools, timers shall be deployed cautiously and only if no another alternative is available. The time consumed to start and stop timers affects the real time execution of test cases especially in TP parallel routing, since the restbus simulation has to control and monitor lots of timing parameters for all applied *TP_Routing_Instances* of the test case. To solve this problem, differences between timestamps are calculated to fulfill most of the timing requirements of the test system, as for example calculating the time gap between two frames to monitor defined timing parameters.

Nevertheless, the usage of timers is non-avoidable, since calculating the differences between timestamps suffers from the problem of infinite waiting loops. For example, if an expected frame has been dropped because of an error, the time difference cannot be calculated and the restbus simulation would enter an infinite waiting loop.

For each *TP_Routing_Instance* of TP parallel routing, the restbus simulation utilizes some timers to observe and control the communication on both source and destination networks. These timers can vary depending on the communication bus (CAN, FlexRay, LIN) and the type of the involved networks (Source or Destination). Required timers are listed below:

- For FlexRay network as a source or destination network: a unique timer to monitor the Cycle Repetition parameter of each simulated FlexRay ECU of TP parallel routing.
- For CAN network as a source network: a unique timer to monitor the minimum separation time (STmin) parameter of each simulated CAN ECU of TP parallel routing.
- Timer for the execution time of the test case: This timer shall terminate the test case execution once the TP routing has encountered an error to avoid infinite waiting loop.

All other timing parameters are monitored and evaluated based on the differences between registered timestamps.

Send Functions of the Restbus Simulation

In order to transmit data and control frames of TP parallel routing, several send functions are implemented in the restbus simulation. These functions are summarized as follows:

- *Send_Single_Frame_CAN_Parallel(int References[])*: to start parallel unsegmented physical TP data routing on a CAN Source Network. The function is able to handle following addressing options in this case: normal addressing with 11 bit CAN identifier, normal fixed addressing with 29 bit CAN identifier, extended addressing with 29 bit CAN identifier, mixed addressing with 11 bit CAN identifier and mixed addressing with 29 bit CAN identifier. This function requires as parameter an integer array with references of *TP_Routing_Instances* of the test case to be executed.
- *Send_Functional_Frame_CAN_Parallel(int References[])*: to start parallel unsegmented functional TP data routing on a CAN Source Network. The function is able to handle following addressing options in this case: normal addressing with 11 bit CAN identifier, normal fixed addressing with 29 bit CAN identifier, extended addressing with 29 bit CAN identifier. This function requires as parameter an integer array with references of *TP_Routing_Instances* of the test case to be executed.
- *Send_First_Frame_CAN_Parallel(int References[])*: to start parallel segmented physical TP data routing on a CAN Source Network. The function is able to handle following addressing options in this case: normal addressing with 11 bit CAN identifier, normal fixed addressing with 29 bit CAN identifier, extended addressing with 29 bit CAN identifier, mixed addressing with 11 bit CAN identifier and mixed addressing with 29 bit CAN identifier. This function requires as parameter an integer array with references of *TP_Routing_Instances* of the test case to be executed.

- *Send_FC_CTS_Frame_CAN(int Reference)*: to send a Flow Control Frame with CTS control information on a CAN Destination Network of TP routing. The function needs as a parameter an integer value which holds the reference of processed *TP_Routing_Instance*.
- *Send_CF_Block_CAN(int Reference)*: to send a block of CAN Consecutive Frames on a CAN Source Network of TP routing. The function requires as a parameter an integer value that holds the reference of processed *TP_Routing_Instance*.
- *Send_Start_Frame_Unsegmented_FR_Parallel(int References[])*: to start parallel unsegmented physical TP data routing on a FlexRay Source Network. The function is able to handle following addressing options in this case (all addressing formats are acknowledged/unacknowledged, known/unknown message length): normal addressing, normal fixed addressing, extended addressing and mixed addressing. The function requires as a parameter an integer array that holds the references of processed *TP_Routing_Instances*.
- *Send_Functional_Frame_FlexRay_Parallel(int References[])*: to start parallel unsegmented functional TP data routing through sending Start Frame on a FlexRay Source Network. The function is able to handle following addressing options in this case (all addressing formats are unacknowledged and with known message length): normal addressing, normal fixed addressing and extended addressing. The function needs as a parameter an integer array which holds the references of processed *TP_Routing_Instances*.
- *Send_Start_Frame_Segmented_FR_Parallel(int References[])*: to start parallel segmented physical TP data routing on a FlexRay Source Network. The function is able to handle following addressing options in this case (all addressing formats are acknowledged/unacknowledged, known/unknown message length): normal addressing, normal fixed addressing, extended addressing and mixed addressing. The function needs as a parameter an integer array which holds the references of processed *TP_Routing_Instances*.
- *Send_FC_CTS_Frame_FR(int Reference)*: to send a Flow Control Frame with CTS control information on a FlexRay Destination Network. This function needs as a parameter an integer value which holds the reference of processed *TP_Routing_Instance*.
- *Send_CF_Block_FR(int Reference)*: to send a block of FlexRay Consecutive Frames on a FlexRay Source Network. This function needs as a parameter an integer value which holds the reference of processed *TP_Routing_Instance*.
- *Send_LF_Frame_FR(int Reference)*: to send a FlexRay Last Frame on a FlexRay Source Network. This function needs as a parameter an integer value which holds the reference of processed *TP_Routing_Instance*.

7.1.5 Analysis

During the execution of a test case for TP parallel routing, the restbus simulation shall process individual *TP_Routing_Instances* correctly. In order to achieve this task, the restbus simulation takes over the following responsibilities:

- It triggers TP data routing on the corresponding Source Networks.
- It evaluates responses of the gateway test object on the corresponding Source and Destination Networks. The evaluation process comprises checking the content of response frames as well as the related timing parameters.
- It sends responses on the Source and Destination Networks as specified in the test case.
- It reports errors and performance information.
- It adapts the test case selection and generation process according to the information gained from run test cases.

To carry out these responsibilities, the restbus simulation makes use of the send functions defined previously. Additionally, it utilizes three different arrays to save information related to the routing behavior of individual *TP_Routing_Instances*. The first array (*TP_Routing_Instances_Information[][]*) is a two dimensional array used to hold the current status of routing during the test case run. The first index of the array is a reference to individual *TP_Routing_Instances* of the test case, while the second index refers to information related to individual *TP_Routing_Instances* of the test case. This information is summarized as below:

- FS, BS and STmin parameters of the last received Flow Control Frame in the case of CAN networks.
- FS, BC and BfS of the last received Flow Control Frame in the case of FlexRay networks.
- SN of the last sent and the last received CAN frame in the case of CAN networks.
- SN of the last sent and the last received FlexRay frame in the case of FlexRay networks.
- The total number of sent bytes on the Source Network.
- The total number of received bytes on the Destination Network.

The second array (*TP_Routing_Instances_Times[][]*) is also a two dimensional array used to hold the timestamps of frames in order to calculate time differences and monitor timing parameters. During test case run, following timestamps are saved for individual *TP_Routing_Instances*:

- Timestamp of the First Frame sent on the Source Network in the case of a CAN network.
- Timestamp of the First Frame routed on the Destination Network in the case of a CAN network.
- Timestamp of the Start Frame sent on the Source Network in the case of a FlexRay network.
- Timestamp of the Start Frame routed on the Destination Network in the case of a FlexRay network.

- Timestamp of the last Consecutive Frame sent on the Source Network.
- Timestamp of the last Consecutive Frame routed on the Destination Network.
- Timestamp of the last Flow Control Frame received on the Source Network.
- Timestamp of the last Flow Control Frame sent on the Destination Network.
- Timestamp of the Last Frame sent on the Source Network in the case of a FlexRay network.
- Timestamp of the Last Frame routed on the Destination Network in the case of a FlexRay network.

The third array (*TP_Routing_Instances_Performance[][]*) is used to collect information related to TP routing performance of the gateway test object. Following TP routing details are saved in this array:

- Maximum and minimum routing time of the First Frames and the Start Frames with references to the corresponding *TP_Routing_Instances* and the utilized test cases.
- Maximum and minimum routing time of Consecutive Frames with references to the corresponding *TP_Routing_Instances* and the utilized test cases.
- Maximum and minimum time to receive a Flow Control Frame from the gateway test object with references to the corresponding *TP_Routing_Instances* and the utilized test cases.
- Maximum and minimum routing time of the whole defined data with references to the corresponding *TP_Routing_Instances* and the utilized test cases.
- Maximum and minimum count of received Flow Control Frames with the status (WAIT) on the Source Network with references to the corresponding *TP_Routing_Instances* and the utilized test cases.
- Maximum and minimum count of Frames saved temporarily in the gateway test object with references to the corresponding test cases.

7.1.6 Reporting

Reporting is an important part of software testing. It is a mechanism to represent test results in a way that helps the customer to understand the status of the product and consequently make the right decision.

Basically, a test report should have enough information about the SUT and its tested functionality. One of the most important features of a test report is the granularity. This must be maintained for different persons reading the report, such as developers and managers, to understand the efficiencies/deficiencies of the system. Furthermore, it is recommended to use

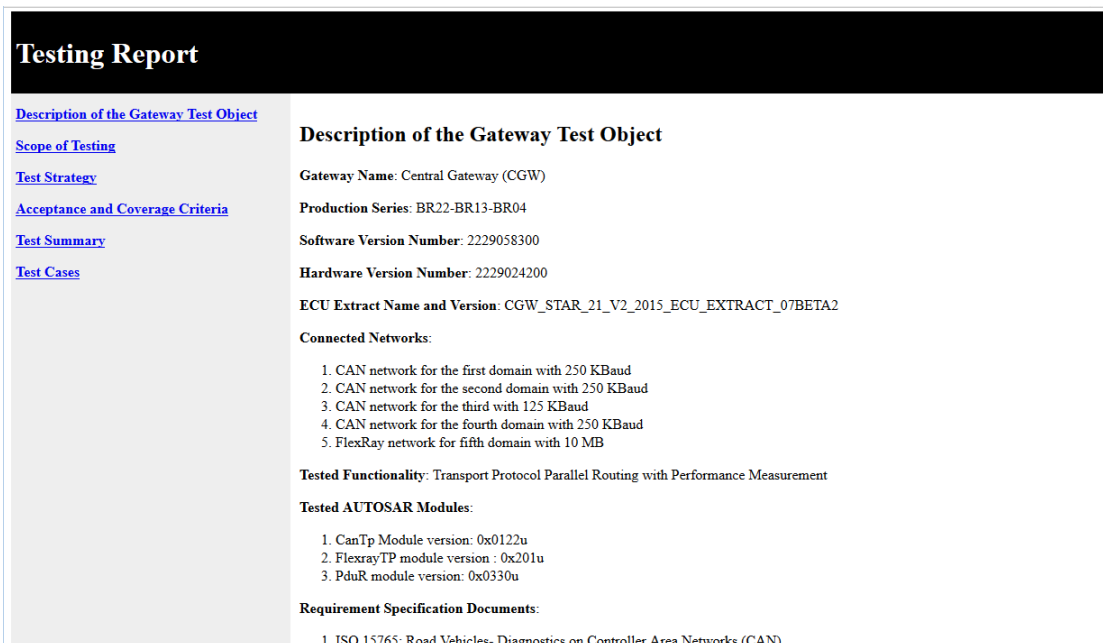


Figure 7.2: HTML Report: Description Section

graphical representation to enhance the readability of the report by utilizing mechanisms such as charts and tables.

In the test system developed in this thesis, the report is designed to include the following sections (see fig. 7.2):

- **Description of the Gateway Test Object**
This section of the main page of the report contains general information about the gateway test object as well as the purpose of testing. It highlights the tested functions and features with a reference to the corresponding requirement specifications. Information included here are listed below:
 - Name and type of the gateway test object.
 - Production series that utilize the gateway test object.
 - Number and characteristics of the connected networks.
 - Software and hardware versions of the gateway test object.
 - Tested functionalities of the gateway test object and information about the involved software modules with their versions.
 - References to the requirement specification documents.
- **Scope of Testing**
In this section, the scope of testing is communicated to the reader. Testing in this thesis has two scopes documented in this section. These are the *functionality* and the *performance*.

Description of the Gateway Test Object	TEST CASE 1 : PASSED
Scope of Testing	
Test Strategy	The Test case combines 33 TP_Routing_Instances. These are:
Acceptance and Coverage Criteria	TP_Routing_Instance_0, 18da58f2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf258 2 8 8 10 -1 -1 0 0 2
Test Summary	TP_Routing_Instance_1, 18da61f2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf261 2 8 8 10 -1 -1 0 0 2
Test Cases	TP_Routing_Instance_2, 18da5cf2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf25c 2 8 8 10 -1 -1 0 0 2
	TP_Routing_Instance_3, 18da5af2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf25a 2 8 8 10 -1 -1 0 0 2
	TP_Routing_Instance_4, 18da5df2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf25d 2 8 8 10 -1 -1 0 0 2
	TP_Routing_Instance_5, 18da29f2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf229 2 8 8 10 -1 -1 0 0 2
	TP_Routing_Instance_6, 18da59f2 8 1 10 8 -1 -1 -1 -1 -1 0 18daf259 2 8 8 10 -1 -1 0 0 2
Execution Details of the Test Case	-1 0 18daf262 2 8 8 10 -1 -1 0 0 2
ECU0_RQ TX CAN_Bus= 1 ID= 18da58f2h T= 1002.3600000ms, 07 01 02 03 04 05 06 07	99 2 8 32 0 -1 -1 0 0 2
ECU1_RQ TX CAN_Bus= 1 ID= 18da61f2h T= 1002.6500000ms, 07 01 02 03 04 05 06 07	98 2 8 32 0 -1 -1 0 0 2
ECU0_RQ_Routed CAN_Bus= 2 ID= 18da58f2h T= 1002.6700000ms, 07 01 02 03 04 05 06 07	494 2 8 32 0 -1 -1 0 0 2
ECU2_RQ TX CAN_Bus= 1 ID= 18da5cf2h T= 1002.9200000ms, 07 01 02 03 04 05 06 07	
ECU1_RQ_Routed CAN_Bus= 2 ID= 18da61f2h T= 1002.9700000ms, 07 01 02 03 04 05 06 07	
ECU3_RQ TX CAN_Bus= 1 ID= 18da5af2h T= 1003.2000000ms, 07 01 02 03 04 05 06 07	
ECU2_RQ_Routed CAN_Bus= 2 ID= 18da5cf1h T= 1003.2700000ms, 07 01 02 03 04 05 06 07	
ECU4_RQ TX CAN_Bus= 1 ID= 18da5df2h T= 1003.4800000ms, 07 01 02 03 04 05 06 07	
ECU3_RQ_Routed CAN_Bus= 2 ID= 18da5af1h T= 1003.5600000ms, 07 01 02 03 04 05 06 07	
ECU5_RQ TX CAN_Bus= 1 ID= 18da29f2h T= 1003.7700000ms, 07 01 02 03 04 05 06 07	
ECU4_RQ_Routed CAN_Bus= 2 ID= 18da5df1h T= 1003.8600000ms, 07 01 02 03 04 05 06 07	
ECU6_RQ TX CAN_Bus= 1 ID= 18da59f2h T= 1004.0500000ms, 07 01 02 03 04 05 06 07	
ECU5_RQ_Routed CAN_Bus= 2 ID= 18da29f2h T= 1004.1500000ms, 07 01 02 03 04 05 06 07	
ECU7_RQ TX CAN_Bus= 1 ID= 18da62f2h T= 1004.3300000ms, 07 01 02 03 04 05 06 07	
ECU6_RQ_Routed CAN_Bus= 2 ID= 18da59f2h T= 1004.4500000ms, 07 01 02 03 04 05 06 07	
ECU8_FF TX CAN_Bus= 1 ID= 4e9h T= 1004.5700000ms, 1f ff 01 02 03 04 05 06	
ECU7_RQ_Routed CAN_Bus= 2 ID= 18da62f2h T= 1004.7400000ms, 07 01 02 03 04 05 06 07	
ECU9_FF TX CAN_Bus= 1 ID= 656h T= 1004.8100000ms, 1f ff 01 02 03 04 05 06	
ECU0_RS TX CAN_Bus= 2 ID= 18daf258h T= 1005.0300000ms, 07 01 02 03 04 05 06 07	
ECU8_Flow_Control_CTS RX CAN_Bus= 1 ID= 499h T= 1005.0400000ms, 30 20 00 aa aa aa aa aa	

Figure 7.3: HTML Report: Test Cases Section

- **Test Strategy**
In this section, the testing strategy is described to help the reader to understand how test cases are selected and generated.
- **Acceptance and Coverage Criteria**
In this section, the test acceptance and coverage criteria are described.
- **Test Summary**
This section addresses on a very abstract level some of the test details like start/end dates, the tools used and their versions, the number of generated test cases, the number of errors and a performance summary of the gateway test object.
- **Test Cases**
In this section, detailed information about the structure of individual test cases and their execution is presented (see fig. 7.3). Hereby, the reader has the ability to navigate through executed test cases to gain information like the involved *TP_Routing_Instances* and the routing performance related to each of them.

7.2 Gateway Test Object

The gateway test object used to carry out the experiments in this thesis consists of a hardware and a software part. The software part is developed on the basis of the AUTOSAR standard version 3.2. The transport protocols of the software are based on the ISO standards, ISO 10681 for FlexRay TP [11], and ISO 15765 for CAN TP [7]. Regarding the hardware part, the evaluation board V850E2/Fx4 with the following features is deployed:

- 32-bit microcontroller.
- 6 configurable serial interfaces for CAN communication protocol.
- 2 configurable serial interfaces for FlexRay communication protocol.
- 6 configurable serial interfaces for LIN communication protocol.
- 6 configurable serial interfaces for RS232 communication protocol.
- LEDs, switches and buttons for direct user interactions.
- Debug and Flash Programming Connectors.

7.3 General Information

This section contains general information related to defined aspects of the proposed combinatorial test approach. This information is required by the test system to apply the approach and generate executable test cases.

7.3.1 TP Routing Scenarios

The determined *TP_Routing_Scenarios* are:

- Flashing (S_1).
- Uploading (S_2).
- OBD in the one direction (S_3).
- OBD in the other direction (S_4).
- Functional TP routing (S_5).

7.3.2 Similarity Criteria

The Similarity criteria used to form groups are:

- Network relationships of *TP_Routing_Channels* (*Source_Network* and *Destination_Network*).
- Addressing format (normal or mixed).
- Block Size (BS) and minimum separation time (STmin) if they are available.
- Cycle Repetition if it is available.
- ID Type (normal or extended).
- Address Type (physical, functional or extended).
- *TP_Routing_Scenario*

7.3.3 Conflicts

The determined conflicts are listed below:

- Type_A-Conflicts
 - Flashing and Uploading are non-combinable
 - Flashing and OBD in both directions are non-combinable
 - Uploading and OBD in both directions are non-combinable
 - OBD in both directions are non-combinable
- Type_B-Conflicts
 - *TP_Routing_Channels* with the same *Request_Identifier* are non-combinable
 - *TP_Routing_Channels* with the same *Request_Slot_ID*, *Base_Cycle* and *Cycle_Repetition* are non-combinable
- Type_C-Conflicts
 - The maximum configured number of parallel *TP_Routing_Instances* for CAN-CAN routing (50 in the first experiment and 10 in the second experiment).
 - The maximum configured number of parallel *TP_Routing_Instances* for CAN-FlexRay routing (10 in the first experiment and not available in the second experiment).
 - The maximum configured buffers for transport protocols (50 in the first experiment and 10 in the second experiment).
 - The maximum configured transmit and receive channels for transport protocols (50 in the first experiment and 10 in the second experiment).

7.4 The First Experiment

The central gateway used in the first experiment is a special electronic control unit which connects five different networks representing five functional domains of a modern vehicle. Bus systems of the five networks are listed as follows:

- A CAN network for the first functional domain with 250 Kilobaud, denoted as bus 1.
- A CAN network for the second functional domain with 250 Kilobaud, denoted as bus 2.
- A CAN network for the third functional domain with 125 Kilobaud, denoted as bus 3.
- A CAN network for the fourth functional domain with 250 Kilobaud, denoted as bus 4.
- A FlexRay network for the fifth functional domain with 10 MB, denoted as bus 5.

The central gateway has 390 configured *TP_Routing_Channels* that can be categorized as follows:

- 190 *TP_Routing_Channels* configured to route TP data from an External Diagnostic Device to available ECUs over the gateway.
- 190 *TP_Routing_Channels* configured to route TP data from available ECUs to an External Diagnostic Device over the gateway.
- 4 *TP_Routing_Channels* configured to route TP data between 4 couples of ECUs in the one direction.
- 4 *TP_Routing_Channels* configured to route TP data between 4 couples of ECUs in the another direction.
- 2 functional *TP_Routing_Channels*.

7.4.1 Results of the Experiment

Based on the previously defined similarity criteria, 39 Extended Groups of similar *TP_Routing_Instances* are built for the central gateway test object. The implementation of the method to handle Type_A-Conflicts among these groups resulted in 4 different sub-IPMs. These sub-IPMs with their corresponding groups are represented in tables 7.1, 7.2, 7.3 and 7.4.

Applying the methodology of SNRs and MNRs to the sub-IPMs, along with the approaches to handle Type_B- and Type_C-Conflicts, led to the following results:

- Four SNRs and one MNR for the first sub-IPM with the following number of combinations:

Table 7.1: First Experiment: First Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (19 elements)	1,2	Mixed	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₂ (8 elements)	1,2	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₃ (14 elements)	1,2	Normal	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₄ (8 elements)	1,2	Normal	(12,10) (12,10)	-	Extended	Physical	S ₁
G ₅ (58 elements)	1,3	Mixed	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₆ (29 elements)	1,3	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₇ (5 elements)	1,3	Mixed	(20,0) (20,20)	-	Normal	Physical	S ₁
G ₈ (3 elements)	1,3	Normal	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₉ (8 elements)	1,4	Normal	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₁₀ (2 elements)	1,4	Normal	(4,10) (4,10)	-	Normal	Physical	S ₁
G ₁₁ (13 elements)	1,4	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₁₂ (5 elements)	1,4	Mixed	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₁₃ (8 elements)	1,5	Normal	(20,0) (-,-)	2	Normal	Physical	S ₁
G ₁₄ (1 elements)	1,5	Normal	(20,0) (-,-)	1	Normal	Physical	S ₁
G ₁₅ (1 elements)	1,5	Mixed	(20,0) (-,-)	4	Normal	Physical	S ₁
G ₁₆ (1 elements)	1,5	Normal	(12,10) (-,-)	1	Extended	Physical	S ₁
G ₁₇ (7 elements)	1,5	Normal	(20,0) (-,-)	4	Normal	Physical	S ₁
G ₁₈ (1 elements)	1,(2,3,4,5)	Normal	(-,-) (-,-)	-	Normal	Functional	S ₅
G ₁₉ (1 elements)	1,(2,5)	Normal	(-,-) (-,-)	-	Extended	Functional	S ₅

- 10 combinations from the first SNR that consists of groups G₁, G₂, G₃, G₄, G₁₈ and G₁₉.
Table 7.5 represents the *Combination Table* for the first SNR of the first sub-IPM after invalid combinations are handled. It is noted in this *Combination Table* that although the group G₁ contains 19 *TP_Routing_Instance*s, only combinations of 5 of them are allowed in parallel due to Type_B-Conflicts.
- 152 combinations from the second SNR that consists of groups G₅, G₆, G₇, G₈ and G₁₈.
- 1 combination from the third SNR that consists of groups G₉, G₁₀, G₁₁, G₁₂ and G₁₈.
- 31 combinations from the fourth SNR that consists of groups G₁₃, G₁₄, G₁₅, G₁₆, G₁₇, G₁₈ and G₁₉.
- 4 combinations from the MNR that consists of Groups G₁, G₅, G₁₂, G₁₇, G₁₈ and G₁₉.
- Four SNRs and one MNR for the second sub-IPM with the following number of combinations:
 - 10 combinations from the first SNR that consists of groups G₁, G₂, G₃, G₄, G₁₇ and G₁₈.
 - 78 combinations from the second SNR that consists of groups G₅, G₆, G₇ and G₁₇.
 - 1 combination from the third SNR that consists of groups G₈, G₉, G₁₀, G₁₁ and G₁₇.

Table 7.2: First Experiment: Second Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (15 elements)	2,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₂ (7 elements)	2,1	Normal	(12,10) (12,0)	-	Normal	Physical	S ₂
G ₃ (19 elements)	2,1	Mixed	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₄ (8 elements)	2,1	Normal	(12,0) (12,0)	-	Extended	Physical	S ₂
G ₅ (63 elements)	3,1	Mixed	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₆ (29 elements)	3,1	Normal	(12,10) (12,0)	-	Normal	Physical	S ₂
G ₇ (3 elements)	3,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₈ (8 elements)	4,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₉ (13 elements)	4,1	Normal	(12,10) (12,0)	-	Normal	Physical	S ₂
G ₁₀ (5 elements)	4,1	Mixed	(12,0) (12,0)	-	Normal	Physical	S ₂
G ₁₁ (2 elements)	4,1	Normal	(4,10) (4,10)	-	Normal	Physical	S ₂
G ₁₂ (8 elements)	5,1	Normal	(-, -) (12,0)	2	Normal	Physical	S ₂
G ₁₃ (1 elements)	5,1	Normal	(-, -) (12,0)	1	Normal	Physical	S ₂
G ₁₄ (7 elements)	5,1	Normal	(-, -) (12,0)	4	Normal	Physical	S ₂
G ₁₅ (1 elements)	5,1	Mixed	(-, -) (12,0)	4	Normal	Physical	S ₂
G ₁₆ (1 elements)	5,1	Normal	(-, -) (12,0)	1	Extended	Physical	S ₂
G ₁₇ (1 elements)	1,(2,3,4,5)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₁₈ (1 elements)	1,(2,5)	Normal	(-, -) (-, -)	-	Extended	Functional	S ₅

Table 7.3: First Experiment: Third Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (3 elements)	3,4	Normal	(6,10) (6,10)	-	Normal	Physical	S ₃
G ₂ (1 elements)	4,5	Normal	(6,20) (-, -)	16	Normal	Physical	S ₃
G ₃ (1 elements)	1,(2,3,4,5)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₄ (1 elements)	1,(2,5)	Normal	(-, -) (-, -)	-	Extended	Functional	S ₅

Table 7.4: First Experiment: Forth Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (3 elements)	4,3	Normal	(6,10) (6,10)	-	Normal	Physical	S ₄
G ₂ (1 elements)	5,4	Normal	(-, -) (12,20)	16	Normal	Physical	S ₄
G ₃ (1 elements)	1,(2,3,4,5)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₄ (1 elements)	1,(2,5)	Normal	(-, -) (-, -)	-	Extended	Functional	S ₅

- 31 combinations from the fourth SNR that consists of groups G₁₂, G₁₃, G₁₄, G₁₅, G₁₆, G₁₇ and G₁₈.
- 10 combinations from the MNR that consists of groups G₃, G₅, G₁₀, G₁₄, G₁₇ and G₁₈.
- Three SNRs and one MNR for the third sub-IPM with the following number of combinations:
 - 1 combination from the first SNR that consists of the group G₁.

Table 7.5: First Experiment: *Combination Table* of the first SNR in the first Sub-IPM

G₁	G₂	G₃	G₄
5	8	14	6
5	8	13	7
5	7	14	7
4	8	14	7
5	8	12	8
5	7	13	8
4	8	13	8
5	6	14	8
4	7	14	8
3	8	14	8

- 1 combination from the second SNR that consists of the group G₂.
 - 1 combination from the third SNR that consists of groups G₃ and G₄.
 - 1 combination from the MNR that consists of groups G₁, G₂, G₃ and G₄.
- Three SNRs and one MNR for the fourth sub-IPM with the following number of combinations:
 - 1 combination from the first SNR that consists of the group G₁.
 - 1 combination from the second SNR that consists of the group G₂.
 - 1 combination from the third SNR that consists of groups G₃ and G₄.
 - 1 combination from the MNR that consists of groups G₁, G₂, G₃ and G₄.

Altogether, 336 combinations were sufficient to measure the performance and verify TP parallel routing of the central gateway in this experiment. Comparing to the number of possible combinations of the starting point, which can be calculated as in (3.18), this is a huge reduction. The test time was around three hours.

Table 7.6 represents an example of a combination from the first SNR of the first sub-IPM. Following numbers of *TP_Routing_Instances* have been selected to generate this combination:

- 5 *TP_Routing_Instances* from group G₁
- 8 *TP_Routing_Instances* from group G₂
- 14 *TP_Routing_Instances* from group G₃
- 3 *TP_Routing_Instances* from group G₄

Table 7.6: First Experiment: An Example of a Resulting Combination

Request ID	Response ID	Network Relationship	BS and STmin	Message DLC	NAD	Addressing Format	ID Type	TP Routing Scenario
0x4e9	0x499	1,2	(20,0) (20,0)	8	14	Mixed	Normal	S ₁
0x4e8	0x498	1,2	(20,0) (20,0)	8	32	Mixed	Normal	S ₁
0x4c4	0x494	1,2	(20,0) (20,0)	8	5	Mixed	Normal	S ₁
0x4d0	0x490	1,2	(20,0) (20,0)	8	8	Mixed	Normal	S ₁
0x4e7	0x497	1,2	(20,0) (20,0)	8	13	Mixed	Normal	S ₁
0x450	0x5d9	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x456	0x5d5	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x7e4	0x7ec	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x625	0x5a5	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x654	0x5d4	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x652	0x5d2	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x624	0x5a4	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x653	0x5d3	1,2	(20,0) (8,10)	8	-	Normal	Normal	S ₁
0x64e	0x5ce	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7e5	0x7ed	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7e1	0x7ea	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x665	0x5e5	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x778	0x788	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x64d	0x5cd	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x656	0x5d6	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x650	0x5d0	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7e2	0x7e8	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7e6	0x7ee	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7d9	0x7e7	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x7e3	0x7eb	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x662	0x5e2	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x65b	0x6db	1,2	(20,0) (20,0)	8	-	Normal	Normal	S ₁
0x18da69f1	0x18daf169	1,2	(12,10) (12,10)	8	-	Normal	Extended	S ₁
0x18da66f1	0x18daf166	1,2	(12,10) (12,10)	8	-	Normal	Extended	S ₁
0x18da43f1	0x18daf143	1,2	(12,10) (12,10)	8	-	Normal	Extended	S ₁

7.5 The Second Experiment

The powertrain controller gateway used in this experiment is a special electronic control unit which connects four different networks representing four functional domains of a modern vehicle. Bus systems of the four networks are listed as follows:

- A CAN network for the first functional domain with 250 Kilobaud, denoted as bus 1.
- A CAN network for the second functional domain with 125 Kilobaud, denoted as bus 2.
- A CAN network for the third functional domain with 500 Kilobaud, denoted as bus 3.
- A CAN network for the fourth functional domain with 250 Kilobaud, denoted as bus 4.

The gateway has 39 configured *TP_Routing_Channels* that can be categorized as follows:

- 18 *TP_Routing_Channels* configured to route TP data from External Diagnostic Device to connected ECUs over the gateway.
- 18 *TP_Routing_Channels* configured to route TP data from available ECUs to External Diagnostic Device over the gateway.
- 3 functional *TP_Routing_Channels*.

7.5.1 Results of the Experiment

Based on the previously defined similarity criteria, 16 Extended Groups of similar *TP_Routing_Instances* are built for the powertrain gateway test object. The implementation of the method to handle Type_A-Conflict among these groups resulted in 2 different sub-IPMs. These sub-IPMs with their corresponding groups are represented in tables 7.7 and 7.8.

Table 7.7: Second Experiment: First Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (5 elements)	1,2	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₂ (2 elements)	1,2	Mixed	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₃ (4 elements)	1,3	Mixed	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₄ (1 elements)	1,3	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₅ (2 elements)	1,3	Normal	(20,0) (20,0)	-	Normal	Physical	S ₁
G ₆ (4 elements)	1,4	Normal	(20,0) (12,10)	-	Normal	Physical	S ₁
G ₇ (1 elements)	1,(2,3,4)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₈ (1 elements)	1,(2,3,4)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₉ (1 elements)	1,4	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅

Table 7.8: Second Experiment: Second Sub-IPM's Groups

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G ₁ (3 elements)	2,1	Normal	(12,10) (12,0)	-	Normal	Physical	S ₁
G ₂ (2 elements)	2,1	Mixed	(12,0) (12,0)	-	Normal	Physical	S ₁
G ₃ (2 elements)	2,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₁
G ₄ (3 elements)	3,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₁
G ₅ (4 elements)	3,1	Mixed	(12,0) (12,0)	-	Normal	Physical	S ₁
G ₆ (2 elements)	4,1	Normal	(12,10) (12,0)	-	Normal	Physical	S ₁
G ₇ (2 elements)	4,1	Normal	(12,0) (12,0)	-	Normal	Physical	S ₁
G ₈ (1 elements)	1,(2,3,4)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₉ (1 elements)	1,(2,3,4)	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅
G ₁₀ (1 elements)	1,4	Normal	(-, -) (-, -)	-	Normal	Functional	S ₅

Since the number of Extended Groups is small in this experiment, the two constructs of SNR and MNR are not employed. That is, the approaches to handle Type_B- and Type_C-Conflicts are

applied directly to the groups of sub-IPMs. This led to the following number of combinations during the selection and generation process:

- 101 combinations from the first sub-IPM.
- 273 combinations from the second sub-IPM.

Altogether, 374 combinations were sufficient to measure the performance and verify TP parallel routing of the powertrain gateway in this experiment. The test time was around three hours.

Summary and Outlook

8.1 Discussion

Depending on the grade of diversity in parameters of *TP_Connection_Channels* and in characteristics of connected networks; the number of formulated groups can increase. The idea is to cover the interactions between formulated groups instead of the interactions between *TP_Routing_Instances*. This helps to avoid similar combinations and contribute to the reduction of the test suit size. If the number N of groups remains high, either a *2-Wise* or *3-wise* coverage criterion can be used.

A drawback of the proposed approach in this thesis is the need of expertise in system functionality in order to define similarity criteria and calculate the *Stress Factor* parameters for the groups. However, this needs to be performed only once. Later on, combinations can be generated and tested automatically for each new release of the system.

8.2 Conclusion and Future Work

In this thesis, a combinatorial test process has been proposed to test the functionality of transport protocol parallel routing of an AUTOSAR gateway system and measure its performance. The proposed approach comprises a methodology to build a creative input parameter model that represents the input space of the system and assists in reducing the number of parameter combinations to be tested. The methodology of input parameter model uses defined similarity criteria to cluster system input parameters represented as *TP_Routing_Instances* into groups whose members stimulate similar behavior.

As conflicts in the resulting input parameter model have to be handled, the two known conflict handling methods *sub-models* and *avoid* are used to prohibit invalid combinations of *TP_Routing_Instances*.

In order to reduce the number of combinations, a novel coverage level has been presented. The proposed coverage level covers the interactions between groups of similar input parameters

instead of the parameters themselves. This helps to avoid similar combinations and increase the test coverage.

For further reduction of the size of the test suite, two optional constructs have been also suggested. The usage of these constructs depends on the number of resulting input similar groups. The first construct SNR is applied to cover simple interactions between network pairs, while the second construct MNR is applied to cover complex interactions between multiple interacting networks.

The proposed approach has been validated and evaluated by means of real industrial examples. One main conclusion of this thesis is that one method or one approach is not sufficient to handle all issues related to combinatorial testing. Once a complex system is considered, diverse problems are emerged. These problems are handled mostly in different research works, where each work focuses merely on one problem to provide a separate solution. However, it is of utmost importance to investigate all approaches suggested in literature to choose a suitable combination of solutions that covers all emerged aspects of the studied system.

Bibliography

- [1] J. Zhan, M. M. Porter, J. Polgar, and B. Vrkljan. Older drivers' opinions of criteria that inform the cars they buy: A focus group study. *Accident Analysis & Prevention*, 61:281–287, 2013.
- [2] M. Broy, S. Kirstan, and B. Schätz. What is the benefit of a model-based design of embedded software systems in the car industry? *In Emerging Technologies for the Evolution and Maintenance of Software Models*, 2011.
- [3] G. Tassey. The economic impacts of inadequate infrastructure for software testing, 2002.
- [4] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Std. 1471-2000*, pages 3–3, 2000.
- [5] A. Sangiovanni-Vincentelli and M. Di Natale. Embedded system design for automotive applications. *IEEE Computer*, 40:42–51, 2007.
- [6] PREEvision. http://vector.com/vi_preevision_en.html, 2015. [Online; accessed 28-May-2015].
- [7] Volcano Network Architect. <http://www.mentor.com/products/vnd/communication-management/vna/>, 2015. [Online; accessed 28-May-2015].
- [8] LIN Specification Package Revision 1.3. *LIN Consortium*, 2002.
- [9] H. Zimmermann. Osi reference model—; the iso model of architecture for open systems interconnection. pages 2–9. Artech House, Inc., Norwood, MA, USA, 1988.
- [10] CAN communication protocol: Release Version 2.0B. http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf, 2015. [Online; accessed 28-May-2015].
- [11] F. Hartwich. Can with flexible data-rate. *Proceedings of the 13th international CAN Conference*, 2012.
- [12] M. Broy, G. Reichart, and L. Rothhardt. Architekturen softwarebasierter Funktionen im Fahrzeug: von den Anforderungen zur Umsetzung. *Springer-Verlag*, 34:42–59, 2011.

- [13] Time-Triggered Ethernet (TTEthernet). <http://www.tttech.com/technologies/ttethernet/>, 2014. [Online; accessed 19-July-2014].
- [14] AUTOSAR Methodology. http://www.autosar.org/fileadmin/files/releases/3-2/methodology-templates/methodology/auxiliary/AUTOSAR_Methodology.pdf, 2014. [Online; accessed 19-July-2014].
- [15] Road vehicles-Diagnostics on Controller Area Networks (CAN)-,. *ISO 15765:2004(E)*, 2004.
- [16] Road vehicles-Communication on FlexRay-,. *ISO 10681:2010(E)*, 2010.
- [17] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Publishers, 1999.
- [18] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. *Fifth Annual IEEE Symposium on eLogic in Computer Science*, pages 428–439, 1990.
- [19] D. Peled. Combining partial order reduction with on-the-fly model-checking. *CAV '94 Proceedings of the 6th International Conference on Computer Aided Verification*, pages 377–390, 1994.
- [20] P. Godefroid. Partial-order methods for the verification of concurrent systems- an approach to the state-explosion problem. *Lecture Notes in Computer Science*, 1032, 1996.
- [21] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. *Lecture Notes in Computer Science*, 1427:147–158, 1998.
- [22] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *ACM*, 31(6):676–686, June 1988.
- [23] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies: A survey. *GMU Technical Report ISE-TR-04-05*, 2004.
- [24] Specification of FlexRay Transport Layer. http://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_FlexRay_TP.pdf, 2014. [Online; accessed 19-July-2014].
- [25] Specification of CAN Transport Layer. http://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_CAN_TP.pdf, 2014. [Online; accessed 19-July-2014].
- [26] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Transaction on Software Engineering*, 23:437–444, 1997.
- [27] R. Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. *Communications of the ACM*, 28:1054–1058, 1985.

- [28] C. J. Colbourn, M. B. Cohen, M. L. Fredman, and R. C. Turban. A Deterministic Density Algorithm for Pairwise Interaction Coverage. *IASTED Intl. Conference on Software Engineering*, pages 345–352, 2004.
- [29] IEEE Standard for Software Unit Testing. *ANSI/IEEE Std. 1008-1987*, 1987.
- [30] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std. 610.12-1990*, 1990.
- [31] G. D. Everett and Raymond McLeod, Jr. *Software Testing: Testing Across the Entire Software Development Life Cycle*. Wiley-IEEE Computer Society Pr, 2007.
- [32] N. Juristo and S. Vegas. *Functional Testing, Structural Testing and Code Reading: What Fault Type Do They Each Detect?* Springer Berlin Heidelberg, 2003.
- [33] Roger S. Pressman. *Software engineering: a practitioner's approach*. McGraw-Hill Higher Education, 7 edition, 2010.
- [34] IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std. 830-1998*, 1998.
- [35] W3CXML. <http://www.w3.org/XML/Schema>, 2014. [Online; accessed 19-July-2014].
- [36] World wide web consortium (w3c). <http://www.w3.org/>, 2014. [Online; accessed 19-July-2015].
- [37] AUTOSAR Specification of the System Template. http://www.autosar.org/fileadmin/files/releases/3-2/methodology-templates/templates/standard/AUTOSAR_SystemTemplate.pdf, 2014. [Online; accessed 19-July-2014].
- [38] J. Bertolino, J. Gao, and E. Marchetti. XML Every-Flavor Testing. *INSTICC Press*, pages 268–273, 2006.
- [39] J. Bing Li and J. Miller. Testing the Semantics of W3C XML Schema. *29th Annual International Computer Software and Applications Conference*, 2:443–448, 2005.
- [40] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Automatic Test Data Generation for XML Schema-based Partition Testing. *Proceeding AST '07 Proceedings of the Second International Workshop on Automation of Software Test*, page 4, 2007.
- [41] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Systematic Generation of XML Instances to Test Complex Software Applications. *Proceeding RISE'06 Proceedings of the 3rd international conference on Rapid integration of software engineering techniques*, pages 114–129, 2007.
- [42] M. Kaur, N. Sharma, and R. K. Kaur. XML Schema Based Approach for Testing of Software Components. *International Journal of Computer Applications*, 6:7–11, 2010.

- [43] J. Guttag. Abstract data types and the development of data structures. *Commun. ACM*, 20(6):396–404, 1977.
- [44] I. Nunes, A. Lopes, V. Vasconcelos, J. Abreu, and L. S. Reis. Checking the conformance of java classes against algebraic specifications. In *In Proceedings of the International Conference Formal Methods and Software Engineering*, volume 4260 of LNCS, pages 494–513. Springer-Verlag, 2006.
- [45] J. J. M. M. Rutten. Universal coalgebra: a theory of systems, 1996.
- [46] J. Gannon, P. McMullin, and R. Hamlet. Data abstraction, implementation, specification, and testing. *ACM Trans. Program. Lang. Syst.*, 3(3):211–223, 1981.
- [47] D. Longuet and M. Aiguier. Specification-based testing for cocasl’s modal specifications. In *Proceedings of the 2Nd International Conference on Algebra and Coalgebra in Computer Science*, pages 356–371, Berlin, Heidelberg, 2007. Springer-Verlag.
- [48] G. Bernot, M. C. Gaudel, and B. Marre. Software testing based on formal specifications: A theory and a tool. *Softw. Eng. J.*, 6:387–405, 1991.
- [49] M. C. Gaudel. Testing can be formal, too. In *Proceedings of the 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 82–96. Springer-Verlag, 1995.
- [50] R. Doong and P. G. Frankl. The astoot approach to testing object-oriented programs. *ACM Trans. Softw. Eng. Methodol.*, 3:101–130, 1994.
- [51] R. Fletcher and A. S. M. Sajeew. A framework for testing object oriented software using formal specifications. In *In Proceedings of the International Conference on Reliable Software Technologies: Ada-Europe, Lecture Notes in Computer Science*, pages 159–170. Springer-Verlag, 1996.
- [52] B. Yu, L. Kong, Y. Zhang, and H. Zhu. Testing java components based on algebraic specifications. *1st International Conference on Software Testing, Verification, and Validation*, pages 190–199, 2008.
- [53] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84:1090–1123, 2002.
- [54] E. Brinksma and J. Tretmans. Modeling and verification of parallel processes. chapter Testing Transition Systems: An Annotated Bibliography, pages 187–195. Springer-Verlag New York, Inc., 2001.
- [55] B. Legeard. Model-based testing: Next generation functional software testing. In *Practical Software Testing : Tool Automation and Human Factors*, Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

- [56] Z. Füredi and R. P. Kurshan. Minimal length test vectors for multiple-fault detection. *Theor. Comput. Sci.*, 315(1):191–208, 2004.
- [57] A. V. Aho, A. T. Dahbura, D. Lee, and M. Uyar. Conformance testing methodologies and architectures for osi protocols. chapter An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours, pages 427–438. IEEE Computer Society Press, 1995.
- [58] J. Tretmans. Formal methods and testing. chapter Model Based Testing with Labelled Transition Systems, pages 1–38. Springer-Verlag, 2008.
- [59] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. Formal methods and testing. chapter Testing Real-time Systems Using UPPAAL, pages 77–117. Springer-Verlag, Berlin, Heidelberg, 2008.
- [60] B. Nielsen and A. Skou. Automated test generation from timed automata. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 343–357, London, UK, UK, 2001. Springer-Verlag.
- [61] L. Frantzen, J. Tretmans, and Willemse. T. A. C. Test generation based on symbolic specifications. In *Formal Approaches to Software Testing*, pages 1–15. Springer Berlin Heidelberg, 2005.
- [62] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 349–364. Springer Berlin Heidelberg, 2005.
- [63] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proceedings of the 9th International Conference on Concurrency Theory*, pages 163–178. Springer-Verlag, 1998.
- [64] L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001.
- [65] R. Groz, O. Charles, and J. Renévo. Relating conformance test coverage to formal specifications. In *FORTE’96*, pages 195–210, 1996.
- [66] L. M. G. Feijs, N. Goga, S. Mauw, and J. Tretmans. Test selection, trace distance and heuristics. In *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, pages 267–282, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.
- [67] C. Jard and T. Jéron. Tgv: Theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Int. J. Softw. Tools Technol. Transf.*, 7(4):297–315, 2005.
- [68] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. *SIGSOFT Softw. Eng. Notes*, 27(4):112–122, 2002.

- [69] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. *SIGSOFT Softw. Eng. Notes*, 29(4):55–64, 2004.
- [70] W. Grieskamp, N. Kicillof, and N. Tillmann. Action machines: a framework for encoding and composing partial behaviors. *International Journal on Software and Knowledge Engineering*, 16:705–726, 2006.
- [71] C. Constant, T. Jéron, H. Marchand, and V. Rusu. Integrating formal verification and conformance testing for reactive systems. *IEEE Trans. Softw. Eng.*, 8(33):558–574, 2007.
- [72] L. Frantzen, J. Tretmans, and T. Willemse. A symbolic framework for model-based testing. *FATES 2006 and RV 2006*, 4262:40–54, 2006.
- [73] K. El-Fakih, N. Yevtushenko, and H. Fouchal. Testing timed finite state machines with guaranteed fault coverage. In: *TestCom 2009*, 5826:66–80, 2009.
- [74] ETSI. 2011b. requirements for modelling notations. *Technical Report ES 202 951*. ETSI, 2011.
- [75] G. J. Myers, T. Badgett, and C. Sandler. *The Art of Software Testing*. Wiley, 2011.
- [76] T. Y. Chen, T. H. Tse, and Y. T. Yu. Proportional sampling strategy: A compendium and some insights. *J. Syst. Softw.*, 58(1):65–81, 2001.
- [77] H. Liu, F. Kuo, and T. Y. Chen. Comparison of adaptive random testing and random testing under various testing and debugging scenarios. *Softw. Pract. Exper.*, 42(8):1055–1074, 2012.
- [78] Huai L., X. Xie, J. Yang, Y. Lu, and T. Y. Chen. Adaptive random testing through test profiles. *Softw. Pract. Exper.*, 41(10):1131–1154, 2011.
- [79] A. Shahbazi, A. F. Tappenden, and J. Miller. Centroidal voronoi tessellations - a new approach to random testing. *IEEE Trans. Softw. Eng.*, 39(2):163–183, 2013.
- [80] A. F. Tappenden and J. Miller. A novel evolutionary approach for adaptive random testing. *IEEE Trans. on Reliability*, pages 619–633, 2009.
- [81] Y. K. Malaiya. Antirandom testing: getting the most out of black-box testing, 1995.
- [82] J. Wegen and O. Buehler. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In *In Proceedings of GECCO*, pages 1400–1412. Springer-Verlag, 2004.
- [83] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Softw. Eng.*, 36(2):226–247, 2010.
- [84] J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Syst.*, 15:275–298, 1998.

- [85] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo. Automated unique input output sequence generation for conformance testing of fsms. *Comput. J.*, 49:331–344, 2006.
- [86] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. Ipog-ipog-d: Efficient test generation for multi-way combinatorial testing. *Software Test., Verif. & Reliab.*, 18:125–148, 2008.
- [87] B. Beizer. *Software Testing Techniques (2nd ed.)*. Van Nostrand Reinhold Co., 2nd edition, 1990.
- [88] K Tai and Y. Lei. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28:109–111, 2002.
- [89] Y. W. Tung and W. S. Aldiwan. Automatic test case generation for the new generation mission software system. *IEEE Aerospace Conference*, 1:431–437, 2000.
- [90] M. Grindal. *Handling Combinatorial Explosion in Software Testing*. PhD thesis, Linköping Studies in Science and Technology, 2007.
- [91] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.
- [92] R. R. Othman, K. Z. Zamli, and S. M. S. Mohamad. T-way testing strategies: A critical survey and analysis. *International Journal of Digital Content Technology and its Application*, 7:222–235, 2013.
- [93] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. *Testing of Communicating Systems IFIP Advances in Information and Communication Technolog*, 48:59–74, 2000.
- [94] A. W. Williams and R. L. Robert. A measure for component interaction test coverage. *IEEE International Conference on Computer Systems and Applications*, 48:304–311, 2001.
- [95] H. Mohammad and S. M. Shamooun. Handling conflicts to test transport protocol’s parallel routing on a vehicle gateway system. *IEEE Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1559–1568, 2014.
- [96] D. R. Kuhn and M. J. Reilly. An investigation of the applicability of design of experiments to software testing. *In Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop*, 13:91–95, 2002.
- [97] D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30:418–421, 2004.
- [98] M. B. Cohen. *Designing Test Suites for Software Interaction Testing*. PhD thesis, University of Auckland, 2004.

- [99] A. W. Williams and R. L. Robert. A practical strategy for testing pair-wise coverage of network interfaces. *In Proceedings of the 7th International Symposium on Software Reliability Engineering*, pages 246–254, 1996.
- [100] M. Grindal, J. Offutt, and J. Mellin. Managing conflicts when using combination strategies to test software. *Software Engineering Conference*, pages 255–264, 2007.
- [101] R. Brownlie, J. Prowse, and M. S. Phadke. Robust testing of at&t pmx/starmail using oats. *AT&T Technical Journal*, pages 41–47, 1992.
- [102] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 13:83–87, 1996.
- [103] J. Czerwonka. Pairwise testing in the real world: practical extensions to test-case scenarios. *In Proceedings of the 24th Pacific Northwest Software Quality Conference*, 2006.
- [104] R. C. Bryce and C. J. Colbourn. A density-based greedy algorithm for higher strength covering arrays. *Softw. Test. Verif. Reliab.*, 19:37–53, 2009.
- [105] Z. Wang, B. Xu, and C. Nie. Greedy heuristic algorithms to generate variable strength combinatorial test suite. *International Conference on, Quality Software*, pages 155–160, 2008.
- [106] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei. Combinatorial methods for event sequence testing. *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 601–609, 2012.
- [107] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling. Augmenting simulated annealing to build interaction test suites. *Proceedings of the 14th International Symposium on Software Reliability Engineering*, pages 394–405, 2003.
- [108] S. A. Ghazi and M. A. Ahmed. Pair-wise test coverage using genetic algorithms. *The 2003 Congress on Evolutionary Computation*, 2:1420–1424, 2003.
- [109] T. Shiba, T. Tsuchiya, and T. Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. pages 72–77. IEEE Computer Society, 2004.
- [110] M. I. Younis and Zamli. K. Z. Mc-mipog: a parallel t-way test generation strategy for multicore systems. *ETRI journal*, 32:73–83, 2010.
- [111] K. Forsberg and H. Mooz. The relationship of system engineering to the project cycle. *INCOSE International Symposium*, 1:57–65, 1991.
- [112] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing. *Proceedings of the 1997 International Conference on Software Engineering*, pages 205–215, 1997.

- [113] M. N. Borazjany, L. S. G. Ghandehari, Y. Lei, R. N. Kacker, and D. R. Kuhn. An Input Space Modeling Methodology for Combinatorial Testing. *IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 372–381, 2013.
- [114] Specification of PDU Router. http://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_PDU_Router.pdf, 2014. [Online; accessed 19-July-2014].
- [115] Specification of LIN Interface. http://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_LIN_Interface.pdf, 2014. [Online; accessed 19-July-2014].

Curriculum Vitae

Hassan Mohammad, born in Syria, started in 2001 his study of Computer Science at the University of Aleppo in Syria and graduated with a bachelor degree in 2006. After finishing his study, he worked as an engineer for one year in a power plant. In 2008, he started a new job as a research assistant at the Tishreen University in Syria.

In 2009, he was a master student at the Ilmenau University of Technology in Germany. After doing his Master thesis at Daimler AG in Sindelfingen, Germany, he received in 2011 his Master Degree in Engineering Informatics from the faculty of Information and Automation in Ilmenau.

From 2012 to 2015, Hassan Mohammad was a researcher and industrial PhD Student at the MBtech Group GmbH & Co. KGaA in Sindelfingen, Germany, in cooperation with the Department of Computer Science and the Fraunhofer Institute for Experimental Software Engineering at the Kaiserslautern University of Technology.