Enhancing Model-Based Development of Embedded Systems

Architecture-Centric Modeling, Simulation and Model-Transformation in an Automotive Context

TAHIR NASEER QURESHI

Doctoral Thesis
Stockholm, Sweden 2012
Academic thesis, which with the approval of the Royal Institute of Technology, will be presented for public review in fulfillment of the requirements for a doctoral thesis in Machine Design. The public review is held in Room Gladen B319, Brinellvägen 85, Kungliga Tekniska högskolan, Stockholm on Friday, 2nd November 2012 at 0900 hours.

© Tahir Naseer Qureshi, September 2012

Print: Universitetsservice US AB
Abstract

The increased usage of embedded computer systems in products like automobiles has not only introduced new innovations, additional safety and comfort but also increased the product and development complexity. Several model-based development (MBD) approaches have been proposed to support the management of such complexity. The thesis is aimed towards an integrated environment for MBD of automotive embedded systems. The envisioned environment features model exchange, and choice of modeling techniques, formalisms and tools in an efficient manner.

The first contribution is an integration of EAST-ADL, an automotive-specific ADL with a timed automata (TA) formalism for verifying embedded systems. The focus is mainly on EAST-ADL’s Timing Model (TM) and Behavior Description Annex (BDA). The TM is used for specifying a system’s timing related constraints such as delays and precedence. The BDA not only provides support for modeling behavior using a common formalism but also combines different behavior types for expressing logic, execution and error. The results are a) a formal interpretation of the TM through its transformation to TA, and b) an algorithm for transforming BDA to TA. While the former enables checking consistency between the artifacts of a TM the latter can be used for a holistic behavioral analysis.

In the second contribution, different possibilities to realize EAST-ADL models by AUTOSAR software architecture (a standard for developing automotive embedded software) are studied. The main result is an enhanced mapping scheme between EAST-ADL and AUTOSAR. The findings can serve as guidelines for realizing configurations in EAST-ADL as AUTOSAR parameters.

The third contribution addresses advanced embedded system features by evaluating the TM and TA for dynamic configuration mechanisms and studying Stateflow and SimEvents as alternatives for simulating architectural specifications based on EAST-ADL’s BDA. The results include a) an account of possibilities and issues related to the TM and TA integration studied in this thesis for dynamic configuration mechanisms, b) a comparison of Stateflow and SimEvents in terms of both underlying modeling formalisms and as tools and c) a discussion on possible future opportunities and issues for integrating EAST-ADL, SimEvents, Stateflow and timed automata for the envisioned integrated development environment.

The work is supported by several case studies including a brake-by-wire system, an emergency braking system, a position and a fuel control system, an automatic drive train, and a dynamic reconfiguration scenario related to the relocation of a software component from a failed processing unit to a working one in a microprocessor-based distributed system.

Keywords: Embedded Systems, Architecture, Behavior, EAST-ADL, Modeling, Simulation, AUTOSAR, Timed Automata, Model Transformation, SimEvents, Stateflow.
Acknowledgments

First of all I would like to thank Prof. Martin Törngren for accepting me as his PhD student and introducing me to the world of embedded systems. Without his supervision, continuous guidance, a lot of valuable time spent on the work related to my research as well as the motion and control exercises during the course of my studies, it would not have been possible at all to reach this milestone of my educational career.

Dr. De-Jiu Chen has been my co-supervisor throughout my PhD studies, not only contributing in supervision but also helping me in enhancing my scientific skills and thinking.

Magnus Persson started his PhD studies at almost the same time as me. He has not only collaborated with me during the initial part of my research work but also helped me in improving my Swedish Language to a great extent. Tack för allt Magnus!

During the course of my PhD studies, I had a lot of interesting and fruitful discussions with other members of the Real-Time Controls group at the Department of Machine Design. These provided me with ideas and a lot of feedback on different parts of my research. This especially includes Carl-Johan Sjöstedt, Lei Feng and Matthias Biehl. Lei has also been one of the co-authors of many of my research publications.

My colleagues at the Mechatronics division and administration have also played a very helpful role during my stay at the department. Special thanks to Bengt Eriksson for his support in the form of software tools. Ahsan Qamar, and Fredrik Asplund for reviewing the early drafts of the thesis. Many thanks to Vicki Derbyshire for the painstaking work of proof-reading a large part of this thesis report.

A lot of people have helped, collaborated and cooperated during my research work in several ways. While a few of the people were partners in the DySCAS, ATEST2 and MAENAD projects, others are the researchers and engineers from all over the world who provided me with their research papers and tools which were not easily and readily accessible.

Finances play an important role in a researchers life. Therefore, I would like to thank my financiers i.e. the Higher Eduction Commission of Pakistan for financing the first four years of my PhD studies and the funding agencies like EU and
VINNOVA for the projects through which my research during the last couple of years was financed.

There are a few people who made very little or no contribution to my research work and this thesis report but do play a very important role in my life. I am grateful to my mother Taban Qureshi and father Mushir Ahmad Qureshi for the care, guidance, support and *timely corrections* throughout my life which led me to the place where I am today.

My sisters have always been very caring and boosted my morale whenever needed. In particular my sister Amber Fishan whose expertise in Mechanical Engineering influenced my choice for a multi-disciplinary field. She also persuaded me to learn Matlab at a time when I did not have a clue what it was and what role it would play for me.

Special thanks to my wife Afifa Rahatulain for her support and patience during the thesis work. Last but not least my son Ayaan for providing me a motivation to finish the thesis as soon as possible.

Thank you all!

Tahir Naseer Qureshi

Stockholm
November 2, 2012
Contents

Acknowledgments v

Contents vii

List of Appended Papers xiii

List of Other Publications xv

1 Introduction 1

1.1 Mechatronics and Embedded Systems 1

1.1.1 Future Embedded Systems 2

1.1.2 Complexity Issues 4

1.1.3 Complexity Management Strategies 5

1.1.4 Technologies for Complexity Management 7

1.2 Theoretical Foundations 8

1.2.1 Embedded System Behavior Classification 8

1.2.2 Languages, Formalisms, and Tools 9

1.2.3 Model Transformation 9

1.2.4 Platform-Based Design 10

1.2.5 Tool Integration 11

1.2.6 Multi-View Modeling and Model Consistency 11

1.3 Discussion 12

1.4 Thesis Outline 12

2 Background - Earlier Work 15

2.1 The DySCAS Project 15

2.2 Results and Findings 16

2.2.1 Requirements for a Self-Managing System 16

2.2.2 Requirements for Modeling and Simulation Support 18
2.2.3 Evaluation of Modeling and Simulation Tools for Dynamic Reconfigurations .................. 18
2.2.4 Additional Observations ........................................ 18
2.3 Discussion .......................................................... 19

3 Thesis Overview ...................................................... 21
3.1 Problem Formulation ................................................. 21
  3.1.1 Thesis Problem Space and Objectives ....................... 23
3.2 Thesis Assumptions and Scope ..................................... 26
3.3 Research Questions .................................................. 27
3.4 Methodology ........................................................ 28
  3.4.1 Case Studies: Motivation and Usage ......................... 29
3.5 Summary of the Results ............................................. 31
3.6 Discussion .......................................................... 32

4 State of the Art and Related Work .................................. 33
4.1 Architecture Description Languages ................................ 33
  4.1.1 EAST-ADL ...................................................... 33
  4.1.2 AADL .......................................................... 39
  4.1.3 AUTOSAR ...................................................... 40
  4.1.4 MARTE ......................................................... 41
4.2 Heterogeneous Modeling and Simulation .......................... 42
  4.2.1 MATLAB / Simulink ......................................... 42
  4.2.2 PtolemyII ..................................................... 43
  4.2.3 TrueTime ..................................................... 43
  4.2.4 Metropolis .................................................... 43
4.3 Timed Automata and Model Checking ............................. 44
4.4 Architectures Supporting Dynamic Reconfigurations ............. 45
4.5 Integration of ADLs with Analytical Formalisms .................. 45
4.6 Modeling and Simulation of Dynamically Configurable Architectures 46
  4.6.1 Middleware Simulations ..................................... 46
  4.6.2 Platform-Based Modeling and Simulations .................... 46
4.7 Efforts for Integrating Discrete Event Formalisms and Tools 47
4.8 Discussion .......................................................... 47

5 EAST-ADL Integration with Timed Automata and its Connection with AUTOSAR ........ 49
5.1 EAST-ADL Timing Model and Timed Automata Integration ....... 49
  5.1.1 Functional Components ...................................... 50
  5.1.2 Rate Transition ................................................ 51
  5.1.3 Timing Constraints ............................................ 52
  5.1.4 Verification .................................................... 53
  5.1.5 Mapping Summary and Discussion ............................ 53
5.2 Behavior Description Annex Integration with Timed Automata .... 54
8.3 Overall Contribution towards the Advancement of Science and
Technology .................................................... 95

9 Future Work and Conclusion 97
9.1 Conclusion .............................................. 98

References 99

Appended paper A
Towards the Integration of UPPAAL for Formal Verifica-
tion of EAST-ADL Timing Constraint Specification 113
A.1 Introduction ............................................. 115
A.2 Related Work ............................................. 116
A.3 EAST-ADL .............................................. 117
A.4 UPPAAL ................................................. 117
A.5 Integration Outlook and Focus ......................... 118
A.6 Research Approach and Case Studies ................. 118
A.7 Transformation Scheme ................................ 120
A.8 Prototype Transformation and Verification Results ... 123
A.9 Discussion .............................................. 125
A.10 Acknowledgments ...................................... 126
References .................................................... 126

Appended paper B
A Timed Automata-Based Method to Analyze EAST-ADL
Timing Constraint Specifications 129
B.1 Introduction ............................................. 131
B.2 EAST-ADL and Timing Extension - Concept and Notations .... 132
B.3 Timed Automata and UPPAAL ......................... 135
B.3.1 UPPAAL ............................................. 135
B.4 EAST-ADL and Timed Automata Relationship .......... 136
B.4.1 Mapping Scheme ..................................... 137
B.4.2 Verification .......................................... 142
B.4.3 Usage Considerations and Limitations .............. 143
B.4.4 Mapping Summary ................................... 143
B.5 Brake-by-Wire Case Study ............................. 144
B.6 Related Work .......................................... 147
B.7 Discussion .............................................. 147
References .................................................... 148
List of Appended Papers


  The work was mainly carried out by Tahir Naseer. De-Jiu Chen supported in modeling the EAST-ADL timing constraints. Magnus Persson provided feedback on the timed automata models as well as corrected different language aspects. Martin Törngren has been the main supervisor of the work.


  The work was mainly carried out by Tahir Naseer. De-Jiu Chen supported in modeling the east-adl timing constraints, Martin Törngren has been the main supervisor of the work.

- **Paper C:** Tahir Naseer Qureshi, De-Jiu Chen, Martin Törngren, *Bridging the Gap Between EAST-ADL Behavior Description Annex and UPPAAAL to Enhance Tool Support for Verifying Cyber-Physical Systems*, Submitted to ICCPS 2013, the 4th ACM/IEEE International Conference on Cyber-Physical Systems.

  The work was mainly carried out by Tahir Naseer. De-Jiu Chen and Martin Törngren supervised the work and provided useful feedback.


  While the work was mainly done by Tahir Naseer, De-Jiu Chen not only provided supervision but also timely expertise and information related to
EAST-ADL. Henrik Lönn’s expertise in both EAST-ADL and AUTOSAR were utilized as feedback. Martin Törngren has been the main supervisor and provider of feedback related to quality as well as language errors.
List of Other Publications

Licentiate Thesis


Conferences / Workshops


List of Other Publications

**Book Chapters and Journal Article**


**Technical Reports**


Chapter 1

Introduction

Embedded systems are becoming increasingly common in everyday life. They exist in a large variety ranging from consumer electronics such as personal digital assistants (PDA) to systems on board an aircraft like inertial guidance systems (INS). An embedded system is heterogeneous in nature with a large number of software and hardware components. Due to the heterogeneity, there exist several kinds of complexities (further elaborated in Section 1.1.2). A model-based development approach emphasizing the use of models for different activities is adopted in many engineering disciplines to deal with complexity. This thesis deals with the model-based development of embedded systems which are part of mechatronic products like an automobile.

A large part of the thesis is concerned with the behavioral characteristics of embedded systems dealing with development activities like modeling and simulation, and related model transformations. The work is interdisciplinary where the terms used can have varying interpretations depending on the technical background of the reader. Therefore, the thesis starts with a discussion providing the context and motivation for the work as well as definitions of key terminologies in Sections 1.1 and 1.2. The chapter ends with a brief discussion and an outline of the thesis in Sections 1.3 and 1.4 respectively.

1.1 Mechatronics and Embedded Systems

The advancement of electronics and computer technology gave rise to a multidisciplinary engineering discipline called mechatronics. The term mechatronics is defined as follows [132]:

"Mechatronics is the synergistic combination of precision mechanical engineering, electronic control and systems thinking in the design of products and manufacturing processes. It relates to the design of systems, devices and products aimed at achieving an optimal balance between basic mechanical structure and its overall control".
Typical examples of mechatronic systems include trains and robots. A mechatronic system usually consists of two main modules namely plant and embedded system [115]. A plant is the dynamics under control. The task of an embedded system is to control the plant through actuators and sensors.

An embedded system in a general sense is a system with one or more specific-purpose computer systems as its components. The users of such products do not think of them as a “computer” in the same way as the general purpose home or office computers [57]. Often an embedded system is required to perform its tasks in a timely manner. Such types of embedded systems are referred to as real-time embedded systems. An embedded system has two major types of components, namely hardware and software.

Hardware corresponds to physical components like a microprocessor, cables and their interconnection. A modern embedded system often consists of a large number of processing units and network interconnections. For example, a modern car can have up to 80 processing units called Electronic Control Units (ECU) connected with each other using one or more communication buses [138]. Microprocessors, Digital Signal Processors (DSP) or Field Programmable Gate Arrays (FPGA) are a few examples of processing units. The choice of the kind of processing unit depends on factors such as functionality and cost of hardware. For example, an FPGA can be used for computationally intensive functions like radar data processing. Similarly, for functions like a controller for a DC motor, a microprocessor can be used.

Software plays an important role in embedded systems. In general, software are “the programs and other operating information used by a computer” [101]. In contrast to general purpose software, the main role of embedded software is to interact with the physical world instead of only transformation of data [74]. An embedded system such as an automobile can have up to 2500 software functions [138]. Embedded software includes application software performing specific tasks like the algorithms / logics for controlling a physical device, real-time operating systems and middleware. A real-time operating system (RTOS) provides services such as scheduling and switching of tasks running on a processor in a timely manner [74]. A middleware is a software layer between an operating system and distributed application software, interacting with each other via a communication network [47].

1.1.1 Future Embedded Systems

Due to the advancement of software and hardware technology, embedded systems are going through a paradigm shift towards becoming “intelligent”. This can be seen in the form of modern mobile phones which not only provide services like basic communication but also personal assistance, navigation and customization for individual needs. One of the key features of many advanced systems is the self-management property. The term ‘Self-management’ refers to the following self-X properties [69]:

1. "Self-configuration" refers to the ability of the system to automatically configure itself to adapt to changes in the environment or system resources.
2. "Self-optimization" refers to the ability of the system to automatically adjust its parameters to optimize performance.
3. "Self-healing" refers to the ability of the system to automatically detect and recover from failures.
4. "Self-assessment" refers to the ability of the system to automatically assess its own performance and health.

These self-X properties enable embedded systems to adapt to changing conditions and maintain optimal performance, making them more robust and reliable.
• **Self-configuration** - related to the adjustments made to enable a new configuration to be supported. Dynamic incorporation of newly discovered devices or software is an example of the adjustments related to self-configuration.

• **Self-optimization** - related to the adjustments for improving performance or efficiency. Changing the quality of service levels of an application for improving overall system performance is an example of such adjustments.

• **Self-healing** - related to the adjustments made to mask or recover from a component or software failure. For example, starting a backup configuration in case of a software failure.

• **Self-protection** - related to the adjustments made to deal with threats, such as dynamically detecting and denying a service attack.

Automotive embedded systems are also envisioned to have similar features [4]. Significant benefits can be achieved with the help of existing technologies by enabling and enhancing the cooperation between user devices like smart phones, embedded devices in the vehicle and collaboration with other services utilizing external wireless networks through which a vehicle passes [4]. The benefits include, but are not limited to the usage of location dependent information such as road and weather conditions for route planning; electric and processing power sharing.
between computing devices; achieving fault-tolerance and easy updates by dynamic allocation and relocation of software.

A few future vehicular scenarios are illustrated in Fig. 1.1. The green square on the vehicle in Fig. 1.1 represents a wireless device which can be used to connect to a manufacturer, nearby vehicles or the infrastructure. While the communication with the manufacturer can be used to receive updates of software, the communication between other vehicles and infrastructure can be utilized to form an ad hoc network in case of emergency, providing assistance to the rescue service personnel. The two yellow arrows represent the possibility of relocation of software in case of fault or for optimizing the overall system performance.

1.1.2 Complexity Issues

The term ‘complexity’ is defined as the state or quality of something being complex [101]. The definition of the word complex considered in this thesis is “Consisting of many different and connected parts” which are not easily analyzable or understandable [101].

Embedded systems are inherently complex [130]. This is especially true for real-time embedded systems which are required to react correctly to environmental changes within defined time limits. For example, in case of an automobile, the brakes are required to be engaged within a very short time (typically a fraction of a second) after the brake pedal is pressed. This is not so trivial to achieve due to many factors. An example of such factors is the possible delays in communication between the components implementing the braking functionality [110]. Furthermore, interference between different features can occur if not designed and implemented properly [67].

The development process and the state of practice involve several engineering disciplines. In addition, several stake-holders ranging from a customer, governmental authorities to a development engineer have different functional and technical concerns. For different design phases, the concerns can be related to the part of a system such as hardware or software, or aspects like safety and reliability.

In many cases, the information transfer between different stakeholders is document-based which implies that information in not formally analyzable by computing machines. The use of a document-based approach not only has risks for misinterpretation of information [32] but also consumes more development time [81].

Another aspect of complexity is the artificial complexity due to the tools (software programs) used during the development. A wide range of tools is used by a variety of experts from different engineering disciplines for various activities. The activities can include but are not limited to modeling, simulation and synthesis. These tools often lack integration with each other and therefore information has to be transferred manually from one tool to another. Such transfer of information is in general prone to errors if not handled properly [114].

A lot of testing is carried out for verification of fulfillment of requirements, validity of a product design and ensuring efficiency, reliability, safety and usability,
Several issues related to timing and communication are discovered during the testing process. Timing errors include late reception of a signal, interface errors such as mismatch of data type between two functions communicating with each other. Possibility of difference in protocols followed by different communicating components is an example of communication errors [78]. Many of the testing activities are carried out late in the development process. The later a fault is discovered the more will be the development time and cost.

### 1.1.3 Complexity Management Strategies

There exist several ways to manage complexities related to embedded systems. A general strategy is to follow a divide and conquer approach. This in turn can imply separation of concerns related to different stakeholders, partitioning a system or its views into small modules and abstractions, etc. The following are the three interrelated concepts related to complexity management followed in this thesis.

**Abstraction and Concretization**

One way to deal with the inherent complexities is to consider a system with varying abstraction levels, i.e., the amount of details at different development stages. A higher level of abstraction corresponds to fewer details and vice versa. The process of abstraction, i.e., hiding details enables ease in comprehensibility, analysis of essential information and an early verification and validation of the product under consideration. On the other hand concretization or refinement is the opposite of abstraction.

To understand the two terms, i.e., abstraction and concretization assume a top-down development approach for an automotive braking system. At the highest abstraction level only the features (properties/ functionality/ traits [123]) such as braking control may be considered without any technical details. At this level analysis can be performed for ensuring that the features considered for development are able to fulfill all the requirements. This can be followed by the process of concretization or refinement to another abstraction level where the details related to the system boundary, type of braking control strategy, model of the plant, i.e., vehicle dynamics, etc. related to vehicle braking control can be added. At this level, analysis like efficiency of control strategy can be performed. The concretized model can further be refined to yet another abstraction level by the addition of details like hardware topology, software architecture along with the allocation of software components implementing the algorithms to hardware components.

**Architecture and Architecting**

Architecting plays an important role in the development of systems having complex architecture. Architecture can refer to many things in the context of embedded systems. According to the ISO 42010 [62] an architecture refers to “fundamental
concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”. Similarly, the term architecting is defined as the “process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system’s life cycle” [62]. An architecture model\(^1\) can comprise of multiple model types such as structure and behavior mentioned earlier. It can be used for various purposes such as cost estimates, quality testing and reusing of design knowledge for products having similar characteristics [88].

The architecting of a product is carried out at different abstraction levels with different views encompassing one or more concerns. A concern is defined as “interest in a system relevant to one or more of its stakeholders” [88]. Structure, behavior, performance and communications are a few examples of concerns related to embedded systems. Abstraction levels, views and viewpoints (conventions used defining specific views) can vary depending on the industrial domain, modeling language and the class of embedded system under consideration.

Models and Model-Based Development

Modeling plays an important role in systems development. While modeling is a task to “model something”, a model is “a simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions” [101].

For embedded systems there exist several types of models depending on the concerns and views. The models used for embedded systems development include but are not limited to requirements (for performing requirements engineering activities), structure (related to structural organization of different artifacts), behavior (related to the operational characteristics of a system or its component) and variability (in terms of product or functional configurations). Models can also be classified based on their usage such as specification, analysis and simulation.

A model-based approach is followed in many engineering disciplines for developing complex systems. The term model-based development (MBD) can have several interpretations. The definition followed in this thesis is the use of formalized models for different development activities [133]. An example of MBD is the realization of different control schemes on embedded platforms using models of various kinds. Analytical models are used for analyzing and predicting control performance. The same models are later subjected to code generation for their implementation on a processing hardware. Model-based development is a common practice in several application domains [35, 42, 139]. If used properly and correctly, computerized models can result in reduced time for development activities.

\(^1\)The term architecture and architecture models are used interchangeably in this thesis
1.1.4 Technologies for Complexity Management

A lot of technologies have been developed for managing complexities following the approaches described in Section 1.1.3. The following are the three main technologies relevant for this thesis.

Architecture Description Languages

As the name indicates an architecture description language (ADL) provides a way to formally describe architecture. An ADL can support the definition of a single or multiple model kinds targeting multiple concerns and viewpoints. ADLs are often supported by automated tooling for specification and analysis of architecture models [62]. ADLs can be either domain specific like EAST-ADL [119] or generic like UML [134]. ADLs like EAST-ADL not only provide support for multi-view modeling but also define relationship between different views.

Model-Driven Engineering

Model-driven engineering (MDE) is a software development methodology which combines two main technologies [114]. The first technology is domain-specific languages like ADLs, specializing structure, behavior and requirements in a particular domain such as automotive or avionics. The second technology is the transformation engines and generators such as Eclipse Modeling Framework (EMF) [39] which analyze different aspects of a model and synthesize different artifacts such as source code or other models [114].

While MDE combines different technologies, there also exist domain-specific model-based development tools supporting modeling, analysis and transformation within one environment. Simulink [140] and SCADE [42] are examples of such tools with a wide industrial usage.

Model Checking

Model checking is an automated technique for verifying finite-state reactive systems [23]. In this technique a system is modeled as state transition graphs and the specifications as propositional logics [23]. A model checker is a tool which traverses through all the possible state paths and outputs the result whether the model meets the specifications with an answer true or with a counter-example. There exist several model checking tools having different specification languages and underlying state-transition graph types for traversing through the state space (i.e. the set of all possible values and combination of states). SPIN [9] and UPPAAL [8] are two examples of model checkers. While SPIN is a generic model-checker, UPPAAL targets verification of real-time systems.
1.2 Theoretical Foundations

This section introduces the key concepts relevant for this thesis.

1.2.1 Embedded System Behavior Classification

Depending on the context, the term behavior is used and interpreted in several different ways. The following are different types of behavior as interpreted in this thesis.

- **Hardware and Software Behavior** - An embedded system can be divided into two parts namely hardware (i.e. the physical components like a sensor or processor) and software (i.e. the functionality realized on the hardware). Traditionally, a hardware behavior is more related to the physical dynamics represented by continuous or discrete-time equations, whereas a software behavior is related to the logics which can be modeled by state machines, data-flow graphs, etc. However, with the emergence of reconfigurable hardware like Field Programmable Gate Arrays (FPGA), the hardware can also be described by discrete logics depending on the level of abstraction under consideration.

- **Logical and Execution Behavior** - A logical behavior is related to the functionality, i.e., the input and output relationship of a system. On the other hand execution behavior is related to the aspects such as the frequency of occurrence, delays between the input and output, etc. The distinction between the execution and logical behavior is commonly made for software components and in architecture modeling.

- **Timed and Untimed Behavior** - A behavior with an explicit notion of timing is called timed behavior and vice versa.

- **Specified and Actual Behavior** - A specified behavior is the expected behavior of a system. It can be represented by plain text or by use of a modeling language such as an ADL (Architecture Description Language). An actual behavior is the implemented or observed behavior of a system.

- **Abstract and Refined Behavior** - These concepts are often related to abstraction levels. Details can be added to an abstract behavior to form a refined behavior. For example, the behavior of a controller at a higher abstraction level may only include the logics whilst at a lower abstraction level can have timing aspects added to the logics.

- **Error and Nominal Behavior** - In the context of safety, a hardware or software behavior can be further divided into nominal (related to normal working) and error (related to the case of error) behavior of a system or its component.
• **Fail-safe, Fault-Tolerant and Fault-Silent Behavior** - While fail-safe is a system property implying that the system will not cause any harm in case of a failure, fail-silence and fault-tolerance are related to the behavior on occurrence of a failure. A fault-tolerant behavior can be related to degradation to continue fundamental operations. In contrast a fail-silent system will not do anything in case of a failure.

### 1.2.2 Languages, Formalisms, and Tools

A *modeling language* provides a structured and formalized way to describe a system. A language generally has a *syntax* and underlying *semantics* [53]. While the *syntax* is a collection of valid vocabulary and sentences, the *semantics* provide meaning to the syntax. The semantics can be denotational representing relations or operational representing “actions taken by some abstract machine” [63]. The semantic model underlying a language is termed *modeling formalism*. An alternative term is that of *‘model of computation’*. A language can have multiple underlying modeling formalisms or even add additional concepts to the underlying formalism. For example, UPPAAL timed automata [8] adds concepts of urgency to the underlying timed automata formalisms [11]. Similarly, two different languages developed for the same purpose can have different formalisms.

A *model of computation* (MoC) can be both timed with an explicit notion of time and untimed. The commonly used MoCs for embedded system design are differential equations, difference equations, finite-state machines, synchronous/reactive models, discrete event models, synchronous and asynchronous message passing, timed communicating sequential process (CSP) and timed automata [18]. Some MoCs can also be used as basis for representing other MoCs. For example, a discrete event model of computation can be used for describing other models of computation [143].

*Tools* are software programs for using different languages and performing various development activities. Different tools provide different kinds of support related to the models under consideration. For example, while some tools are used only for specification with or without graphical representation, some others can be used for simulations for analyzing a system behavior based on the underlying modeling formalisms and simulation engine.

Formalisms can also be related to views (encompassing one or more concerns). A viewpoint is supported by one or more formalisms. For example, a software design view can include state machines, data-flow and discrete event formalisms [17].

### 1.2.3 Model Transformation

A literature study on model transformation is presented in [13]. This section provides a summary of the study as an introduction to model transformation.
Model transformation is referred to as the automatic generation of one or more target models from one or more source models [82]. A model transformation can be both unidirectional and bidirectional. Model transformations can be used for:

1. *Synthesis* related to refinement from a higher abstraction level to a lower abstraction level. Code generation from analysis models is a special case of synthesis.

2. *Tool integration* for exchanging data between different tools.

3. *Analysis and simulation*. For example, analyzing safety based on architectural specifications [14].

For automated model transformation, the transformation algorithm or scheme is specified using different transformation languages. The languages can be classified into several types based on the followed transformation method. A couple of the classes relevant for this thesis are:

- **Imperative languages** with focus on operations or sequential control flow of transformations. In other words *how* the transformation should occur [82].

- **Declarative languages** focusing on relations or *what* should be mapped.

- **Hybrid languages** having support for declarative and imperative model transformation.

There exist a lot of transformation languages which are being used by several researchers. The *model query language* (MQL) implemented in MDWorkbench [116] and MetaEdit+ Reporting Language (MERL) [84] are the two languages considered for the thesis. Both MQL and MERL are imperative languages.

An important term related to model transformation is *meta-model*. The term is usually used in the context of Meta-Object Facility (MOF) [90], a standard for model-driven engineering. A meta-model describes the syntax and semantics of a modeling language [82]. Similarly both source and target models conform to their meta-models. The meta-models can be same or different for source and target meta-models depending on the context in which a transformation is developed. Furthermore, a transformation is defined between meta-models instead of models enabling reusability and automation of transformations.

### 1.2.4 Platform-Based Design

Platform-based design [111] is a concept where an application is designed with consideration of the underlying platform, e.g., microprocessors or FPGAs. In this method the platform is abstracted at a higher level for activities like analysis. For example, in its simplest a communication network between two functions can be modeled as a timed delay modeled by a state machine or some other modeling
1.2. Theoretical Foundations

formalism. The platform-based design method is consistent with the three common approaches, i.e., bottom-up, top-down and meet-in-the-middle. There exist several platform-based design methods and tools such as Metropolis [31].

1.2.5 Tool Integration

Tool integration is not directly addressed by this thesis. However, an introduction to the concept and types of tool integration is necessary to understand the applicability of the thesis contribution. The term tool integration is referred to as the process of developing “complete environments that support the entire software development life cycle” [137]. According to Wasserman [137], the following are the types of tool integration [131]:

- **Platform integration**, related to interoperability between one computing machine to another in a distributed environment.

- **Presentation integration**, related to the user experience. Such kind of integration is used for improving the efficiency and effectiveness of user interactions.

- **Data integration**, related to the sharing of data between different tools. The purpose of data integration is to ensure data consistency.

- **Control integration**, related to the control flow between the tools in the form of mechanisms like event notifications. Control integration is intended for flexible communication between different tools according to an underlying process.

- **Process integration**, related to the tools which enable definition and tracking of development activities. The goal of such integration is to ensure an effective interaction between the tools used in a particular development process.

Tool integration is related to model transformation discussed in Section 1.2.3 in the sense that model transformation can be used to integrate different tools in particular for data integration.

1.2.6 Multi-View Modeling and Model Consistency

The concept of multi-view modeling is closely related to terms already described in this chapter such as architecture, models, etc. In its simplest form multi-view modeling refers to the development of different models for various views and activities [136]. The models representing different views are required to be consistent with each other. The term ‘consistent’ as defined in [101] is “not containing any logical contradictions”. In the context of architectures for embedded systems consistency can be considered between a model and a system or between different models [12]. Consistency can be classified in several ways. Some of them are as follows:
• **Weak and strong consistency** [12]: This is related to the scale of connection between the models of a system. The requirement for weak consistency is that every architectural artifact, i.e., components and connectors of a system and every connection between different views have been considered. On the other hand strong consistency also adds a constraint that every view should consider all the artifacts, i.e., components and connections.

• **Vertical and horizontal consistency** [20]: Vertical consistency is related to consistency between models at different development phases or abstraction levels. Horizontal consistency on the other hand refers to the consistency between models at the same phase representing different views.

• **Global and local consistency**: Global consistency is considered from a set of global constraints or relationships. An example of an inconsistent model is a cycle in an inheritance chain between different views of a system represented by UML class diagrams [34]. A locally consistent model can refer to a model whose artifacts are consistent with each other.

Different types of consistency can be achieved in several ways. One way is through the use of architectural description languages (ADL). Several ADLs support different types of consistencies with varying extent. Another way to achieve consistency is through tool integration by providing weak or strong links between different models. This thesis addresses vertical, horizontal and local consistency related to EAST-ADL described in Section 4.1.1.

1.3 Discussion

The chapter introduced a generic context, i.e., embedded systems, related complexities and fundamental concepts relevant for the thesis. Embedded systems due to their heterogeneity are inherently complex. Artificial complexity is added due to lack of integration between a large number of tools being used for managing different aspects of complexity. Several solutions like architecture description languages and model-driven engineering technologies have been introduced. An efficient utilization of these solutions is a need as well as one of the challenges faced by the industry. This need and challenge is the major motivation for this thesis.

1.4 Thesis Outline

The thesis is organized as follows. In chapter 2, the author’s earlier work is presented as a background. This is followed by an overview of the thesis in Chapter 3. An overview of the state of the art technologies and an account of the work related to the thesis are presented in Chapter 4. The results from the appended papers are summarized in Chapter 5. An extension of the work is presented in Chapters 6 and 7. An evaluation of the thesis with respect to the posed
research questions, limitations, validation and additional observations is presented in Chapter 8. The thesis is finally concluded with a brief discussion of the future work and a few concluding remarks in Chapter 9.
Chapter 2

Background - Earlier Work

This thesis has its roots originating from the DySCAS project [37] and an earlier thesis by the author [103]. This chapter introduces the DySCAS project and the result and findings from the author’s work. The results also serve as a basis for this thesis.

2.1 The DySCAS Project

DySCAS (Dynamically Self-Configuring Automotive Systems) [37] was a European Union funded project. The duration of the project was from June 2006 to June 2009. It aimed to develop a middleware technology for future vehicular systems. The vision for the project was self-management in automotive systems as discussed in Section 1.1.1. Several use case scenarios were specified for the project. The scenarios are classified into generic and specific use cases. While a Specific Use Case (SUC) defines a concrete scenario, a Generic Use Case (GUC) describes the common functionality of a combination of several SUCs [25]. The generic use case (GUC) scenarios are as follows:

1. **GUC1: A new device attached to the vehicle** related to the discovery and incorporation of new devices in the vehicle’s communication range.

2. **GUC2: Integrating new software functionality** related to maintaining the software functionalities of both the operating system and application software. This also includes the addition of completely new software which is not known at the design time.

3. **GUC3: Closed reconfiguration** related to the provision of the support for advanced error handling and fault tolerance. The support can include but is not limited to graceful degradation by shutting non-safety critical functions down in case of power shortage, relocation of software in case of failure of a processor, etc.
4. **GUC4: Resource optimization** which is similar to GUC3 but more related to optimization of system resources by balancing of workloads on different ECUs, selection of different scheduling and quality of service techniques for guaranteed quality of service and reliability.

The main outcome of the project was the specifications of the DySCAS middleware and guidelines to develop algorithms and policy for the DySCAS middleware [27]. In addition, reference implementations were also developed for demonstration purpose [26].

### 2.2 Results and Findings

In [103, 104, 106] the author’s work on modeling and simulation of a few DySCAS scenarios is presented. The work was used for validating the development of the DySCAS middleware specifications, demonstration of concepts as well as guidelines for modeling [27]. The results included:

- An identification of requirements for tools to model and simulate self-managing systems.

- An evaluation of Simulink based tools, i.e., TrueTime [92], SimEvents [58] and Stateflow [60] for their support for dynamic configurations [104].

- A primitive mapping scheme between UML [134] and SimEvents/Stateflow in the context of DySCAS middleware development [104].

- A literature study of dynamic load balancing mechanisms [99].

The findings which are relevant for this thesis are as follows. For details the readers are referred to [103].

#### 2.2.1 Requirements for a Self-Managing System

A self-managing system is required to support the following key characteristics:

- **Context awareness:** The system should be aware of architectural design context and system execution context of configuration manipulations. Vehicle location in case of automotive systems, attached/detached devices, system deployment which includes but not limited to processor utilization, network bandwidth and system states including information such as software versions, error codes and policies are a few examples of the context information.

- **Identification of unexpected events:** Detection and identification of component malfunctions, resource imbalance, availability of new software and devices, detachment of devices.
2.2. Results and Findings

- **Run-time verification and change management:** Dynamic reconfiguration in a system implies some kind of intelligence and planning capabilities. Usage of different kind of policies, dynamic and feedback based algorithms enable run-time optimization and evaluation of different configurations. Checking authenticity of a new software, dynamic and runtime decisions related to attachment of a new device or shutting an already existing one off are a few examples of the requirements on verification and decision.

- **Execution of reconfiguration:** A self-configuring system is required to support the following activities:
  
  - Setting up of communication and data routing between the system and a remote device which may include a server at the vehicle manufacturer or workshop for updating and downloading new software.
  - Controlling the power, memory and processor utilization and other resources.
  - Download, installation, rollback, execution and migration of software between different nodes of a distributed system.

All of the above activities are required to be performed in a timely manner which can vary from one system to another. Moreover, these activities should not cause any kind of disturbances or distraction for the system users. These systems should also be robust towards disturbances, errors and failures.

<table>
<thead>
<tr>
<th>Category</th>
<th>Aspect</th>
<th>Property</th>
<th>Description / Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Middleware Component</td>
<td>Structure</td>
<td>Interfaces, data, signals and composition of logics computation means</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Behavior</td>
<td>Execution, error, modes, states, computations.</td>
</tr>
<tr>
<td></td>
<td>Middleware Architecture</td>
<td>Structure</td>
<td>Layering, interfaces, topologies, allocations and composition alternatives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Behavior</td>
<td>Component interactions, startup, actuation of dynamic configurations, sensing of external parameters</td>
</tr>
<tr>
<td>Applications and Platform</td>
<td></td>
<td>Real-time properties</td>
<td>Execution time, end to end delays</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing</td>
<td>Fault injection, configuration changes, failure detection, performance.</td>
</tr>
<tr>
<td>System Quality and Analysis</td>
<td>Levels of Abstraction</td>
<td></td>
<td>Support for different levels of abstraction and a combination of few.</td>
</tr>
<tr>
<td>Setup</td>
<td>Design flow and Reuse</td>
<td></td>
<td>Connection with other tools such as design specification and safety analysis</td>
</tr>
<tr>
<td></td>
<td>Visualization and Comprehensibility</td>
<td></td>
<td>Graphical user interface, hierarchical decomposition.</td>
</tr>
</tbody>
</table>

Figure 2.1: Requirements for modeling and simulation
2.2.2 Requirements for Modeling and Simulation Support

The requirements identified for modeling and simulation are shown in Fig. 2.1. These requirements are detailed in [103, 104].

2.2.3 Evaluation of Modeling and Simulation Tools for Dynamic Reconfigurations

TrueTime and SimEvents (and its combination with Stateflow) were the two tools evaluated for their support to model and simulate dynamic configuration. The evaluation was carried out for modeling and simulation of middleware, operating systems, network protocols and real-time properties, supported levels of abstraction and usability in terms of visualization and comprehensibility of models. The evaluation result is summarized in Fig. 2.2.

<table>
<thead>
<tr>
<th>Evaluated aspects</th>
<th>TrueTime</th>
<th>SimEvents®</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application and middleware</td>
<td>A combination of several tasks.</td>
<td>A subsystem with a combination of Stateflow® / Simulink® / SimEvents™ blocks.</td>
</tr>
<tr>
<td>Operating system</td>
<td>Basic support available as TrueTime kernel.</td>
<td>No direct support but implementable.</td>
</tr>
<tr>
<td>Network</td>
<td>Basic support such as transfer of data, network speed and packet losses.</td>
<td>No direct support but implementable.</td>
</tr>
<tr>
<td>Visualization and comprehensibility</td>
<td>Monitors such as network and cpu schedule are supported but require a lot of Matlab / C code.</td>
<td>Visualization support from Matlab® as well as additional support for entities and attributes. Better visualization due to graphical blocks.</td>
</tr>
<tr>
<td>Real-time properties</td>
<td>Task execution time.</td>
<td>Timing in terms of time-outs, server execution time.</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>Closer to real-time implementation</td>
<td>More suitable for functional analysis.</td>
</tr>
<tr>
<td>Missing aspects</td>
<td>Dynamic linking and loading, state changes.</td>
<td>Dynamic linking and loading, code generation</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Extensible</td>
<td>Extensible</td>
</tr>
</tbody>
</table>

Figure 2.2: Tool evaluation summary

2.2.4 Additional Observations

In addition to the results discussed earlier and in [103], the following observations are made.

- The main difference between the modern systems and future systems is the extent of dynamic configurations and related design. For future systems some aspects like variability are not only required to be considered at design time but also at run time. This further complicates the design and analysis activities for a dynamic reconfigurable system.
• A DySCAS middleware is specified using UML. Being a middleware for automotive systems, it is desirable to know the extent to which existing modeling technologies like EAST-ADL can support DySCAS system specifications.

• Modes, state-machine type of behavior and computations resulting as reaction to an event are common for both existing and future vehicular systems.

• There is a need for an integration of the tools used for specifying the architecture and the tools for analysis. Such integration can be used for bringing life to the specification model by making them executable and the behavior observable.

• Dynamic configuration mechanisms and embedded systems in general require modeling of aspects such as modes and related behavior, need for analysis of real-time properties like end-to-end timing between different components, need for analysis using techniques like fault injection, etc.

• It is non-trivial to graphically model aspects like dynamic linking and loading with the considered tools. For example, in Simulink a subsystem cannot be added to a model at run-time. One way to handle this issue is by an addition of maximum number of possible components in the model. The components can then be enabled / disabled during run-time.

2.3 Discussion

This chapter introduced the DySCAS project and the author’s contribution and findings. The experiences from the DySCAS work have been utilized in the part of thesis presented in Chapters 6 and 7. This especially includes the simulations carried out using Stateflow and Simulink. In [103], four major directions were identified as future work. These include further verification and validation of simulation and analysis models, extension of SimEvents and TrueTime with desirable features listed in Fig. 2.2, model transformation for an enhanced development process and deriving new and enhanced dynamic configuration algorithms and strategies. This thesis focuses on further work on simulation and analysis models and model transformation. The next chapter presents an overview of the thesis followed by the results in the later chapters.
Chapter 3

Thesis Overview

This chapter provides an overview of the thesis. The research problem and related issues addressed by this thesis are discussed in Section 3.1 and, the thesis assumptions and scope in Section 3.2. Sections 3.3 and 3.4 present the research questions and the methodology. A summary of the findings are discussed in Section 3.5 followed by a chapter summary in Section 3.6.

3.1 Problem Formulation

As mentioned in Chapter 1, a model-based approach is useful for developing complex heterogeneous systems since it provides abilities for abstraction and to provide dedicated views. The state of practice employs several model-based development techniques and tools including but not limited to the following [20, 78, 113, 133]:

- Modeling capabilities such as those provided by MATLAB [139] enabling automation and use of advanced tool support.

- Verification using techniques like model-based testing [100] and model checking [23].

- Information management support provided by various ADLs for different development activities at varying abstraction levels in an automated fashion.

- Definition of domain-specific architectures using standard platforms like AUTOSAR [6].

- Automated model synthesis defined in Section 1.2.3 and code generation.

Current model-based approaches are limited in their support for integrating different models/views. Many of the techniques and tools are isolated from each other or have weak integration. This requires manual handling of information,
resulting in issues like loss of information, inconsistencies between different models, increased development time and cost [20, 78]. For example, EAST-ADL [119], an automotive-specific ADL, provides support for specifying architecture at multiple abstraction levels with separation of concerns in a modular manner. On the other hand, tools like Simulink provide support for analyzing system functionalities. Although there is a structural integration [15] between EAST-ADL and Simulink, the behavioral information is manually transferred. This can result in possible inconsistency between Simulink models (corresponding to various parts of an EAST-ADL model) developed by different designers. In addition to achieving consistency, there is a need for reasoning about system properties based on different views and performing trade-off analysis to choose between different architecture configurations and increased safety, reliability, etc.

To avoid the above mentioned issues, seamless integration of these techniques and tools is required [20, 114], constituting one of the current pressing industrial needs to reduce development time and cost [20, 125]. An example of required integration is presented in [14] where a specification is integrated with HIP-HOPS [96] (a safety analysis tool) using EAST-ADL [119] based specifications. The existing isolation of techniques and tools and the required integration is illustrated in Fig. 3.1.

![Figure 3.1: Existing and required technological landscape](image)

The required landscape in Fig. 3.1 illustrates several features. These include but are not limited to:

- The functionalities and requirements are described using different types of models. The models can be used for analysis or specification.
3.1. Problem Formulation

- Associated with models are corresponding languages and tools that describe them. Typically the tools used for describing requirements like MS Word and DOORS [56] are separate from the tools for behavioral analysis like MATLAB.

- Abstraction levels is another building block of Fig. 3.1. The levels could be promoted by guidelines/methodology, and/or be explicitly supported by technologies like an architecture description language.

- Formal methods can be applied on different models.

- The development should be in accordance with industrial standards like ISO 26262 and AUTOSAR.

- All of the above should be applicable for existing as well as advanced functionalities envisioned for future embedded systems.

A seamless integration is challenged by many factors. The factors related to the thesis are as follows:

- Weak semantics: The semantics of many modeling languages like UML are insufficiently defined resulting in different implementations and interpretations [19].

- Gaps between languages: Models specifying the same system can differ due to differences in the modeling languages [20]. For example, some artifacts like events of a Stateflow model specified in Simulink, are discarded when transformed to SCADE state machine models [42].

- Consistency issues: Several kind of consistencies are discussed in Section 1.2.6. The thesis deals with vertical, horizontal and local consistency. The management of consistency is a challenging task, especially if the transformation between models is carried out manually [12, 20].

3.1.1 Thesis Problem Space and Objectives

The landscape discussed earlier is large and is being addressed by researchers all around the world. The problem space considered for this thesis is shown in Fig. 3.2. An architecture description language can be used to specify characteristics like behavior, structure, constraints on behavior and requirements of a system with different views at varying abstraction levels. The architectural models need to be analyzed, which can be done in several ways. Two ways shown in Fig. 3.2 are transforming architectural models to the tools supporting formal methods (arrow 1), and modeling and simulation tools (arrow 4).

Several modeling and simulation tools provide similar support but can have different underlying formalisms and languages. The same applies for formal methods and tools. Therefore it is useful to study the correspondence between different methods, languages and tools (arrow 3). Such kind of study can support
in finding the extent to which different formalisms and tools can be used as alternatives for analyzing architectural specifications. Furthermore, models at different abstraction levels are required to be consistent to ensure a high degree of correspondence between them. One possible way to achieve consistency is by automating the synthesis of models at a lower abstraction level from models at a higher abstraction level.

All the relationships shown by the four arrows in Fig. 3.2 require bi-directionality which can exist in several ways. One example of a bidirectional relationship is the transformation of architecture models to analysis models for formal analysis in one direction and the transfer of results back to architectural information in another direction. The following are the targeted aspects, listed in an order of decreasing focus in the thesis:

1. Integration of behavior models specified using ADLs with analysis models for formal analysis, and its applicability for advanced embedded system functionalities. This is shown by arrow 1 in Fig. 3.2.

2. Model correspondence between design models and automatically synthesized implementation models. This is shown by arrow 2 in Fig. 3.2.
3.1. Problem Formulation

3. Correspondence between different modeling and simulation tools for their usage as alternatives to analyze architecture models and their connection to ADLs. This is shown by arrow 3 in Fig. 3.2.

4. Integration of behavior models specified using ADLs with modeling and simulation tools. This is shown by arrow 4 in Fig. 3.2. This part is not directly addressed. Instead, the experiences from the first three aspects together with the earlier work discussed in Chapter 2 are utilized in this regard.

An embedded system can be both static and dynamic in terms of its configurations. As mentioned in Section 1.1.1, self-management is one of envisioned feature for future embedded systems. Dynamic configuration is one of the key enabling characteristics to achieve self-management. Therefore, the aspects addressed in this thesis should be applicable to both static and dynamic configurations.

The need for integration of techniques and tools can be dealt with in several ways. Guidelines for using specific tools, languages and techniques can be specified. The guidelines can be in the form of work processes consisting of ordering of the work tasks, required checking and synchronization, etc. The actual work for ensuring consistency can be performed manually or in an automated manner. Development of interfaces between different tools and mappings between different modeling languages are prerequisites for such automation. The interfaces can be used for activities like co-simulation or data transfer. Different types of integration related to tools is discussed in Section 1.2.5.

As discussed in Section 1.2.2, different techniques and tools are based on languages whose semantics are defined by explicit or implicit modeling formalisms. Depending on the need and type of integration (discussed in Section 1.2.5), a relationship between different modeling formalisms might need to be established. For the cases where the underlying formalism between different technologies is the same, the relationship is straightforward. In case of different or implicit formalisms, it is more difficult to derive a relationship.

With the four specified problem aspects discussed in this section, the main objectives are to:

- **Objective 1:** Facilitate architecture-centric analysis support for automotive embedded systems by integrating architectural specification models with analysis models on which formal methods can be applied.

- **Objective 2:** Support automated synthesis of EAST-ADL specifications by investigating the relationships between EAST-ADL and the AUTOSAR standard.

- **Objective 3:** Study the relationship between different discrete event formalisms and tools for their usage as alternatives for analyzing specifications modeled using architectural description languages.
3.2 Thesis Assumptions and Scope

The following statements form the basic assumptions for this thesis:

- Model-based development is a key enabling technology for managing complexities related to embedded systems [113, 133].

- An architecture-centric approach placing architecture at the center of the development is more beneficial as opposed to a traditional project-centric approach [16]. It “weaves together product, technology, process, and people into a cohesive lightweight, scalable development method” [73].

- EAST-ADL provides support for managing engineering information related to automotive embedded system architectures. [78].

- A wide industrial usage of ADLs is hindered by their limited tool support [7].

- Formal methods, especially timed automata, can be utilized to verify embedded system architectures [8, 85].

- There is a need to integrate AUTOSAR with architecture description languages [20, 50].

- Dynamic configuration is one of the major characteristics of advanced automotive applications envisioned for future [4, 49].

- Reactive behavior comprising of modes, state-machines and computations on occurrence of an event is common for both existing and future vehicular systems (Section 2.2.4).

- Discrete event modeling formalisms can be used at multiple abstraction levels and for modeling other formalisms like discrete time or differential equations [135, 143].

The coverage and the contributions of the thesis are illustrated in Fig.3.3. The rectangles show the technologies under consideration and the circle connectors show the gaps addressed by the thesis.

The thesis focuses on embedded system behavior from a control systems engineering perspective where the selected industrial domain is automotive. EAST-ADL [119] is the main specification language considered due to a broad coverage of the automotive system development process and model-based information management support. The focus is on the Timing Model (TM) and Behavior Description Annex (BDA) of EAST-ADL.

For formal methods, the thesis only considers timed automata [11] with UPPAAL [8], a timed automata-based model checking tool for analyzing real-time systems as the selected tool. This is related to Objective 1 in Section 3.1.1.

For Objective 2, the work is limited to execution events and structural properties related to the AUTOSAR software architecture.
3.3 Research Questions

For the third objective, the extensions of Simulink namely Stateflow and SimEvents are the considered formalisms and tools. SimEvents and Stateflow not only support two different kinds of discrete event formalisms, but also represent two different types of tools for embedded system design and analysis. Furthermore, Simulink is one of the most common tools used in many industries. SimEvents is a relatively new extension to Simulink providing discrete event simulation support.

In addition to above, the dynamic configuration is limited to a) the scenario of relocation of a software from a failed node to a working one and b) mode related behavior. The latter is common for both static and dynamic configurations.

3.3 Research Questions

With the problem illustrated in Fig. 3.2, scope defined in Section 3.2 and the coverage illustrated in Fig. 3.3, the following research questions are put forth to reach the objectives:

- **Research Question 1:** How can we integrate EAST-ADL with tools based on timed automata for a useful analysis of architectural specifications?

  - **Question 1a:** What are suitable syntactical and semantical mappings between timed automata and EAST-ADL’s timing model and behavior description annex?

  - **Question 1b:** Given a suitable mapping, is it possible to analyze an EAST-ADL timing model for consistency between its artifacts?
– **Question 1c:** Given a solution to Questions 1a and 1b, how can the solution be used for dynamic configuration mechanisms?

- **Research Question 2:** How are AUTOSAR and EAST-ADL related to each other from a behavioral point of view?
  
  – **Question 2a:** What are the possible relationships between EAST-ADL and AUTOSAR architectural configurations?
  
  – **Question 2b:** What are the possible concerns which need to be addressed while refining an EAST-ADL configuration to AUTOSAR?

- **Research Question 3:** How can the modeling formalisms related to Stateflow and SimEvents be used as an alternative for analyzing EAST-ADL based architectural specifications? This in turn implies the need for comparing the two languages.

### 3.4 Methodology

The work in this thesis is carried out as a part of two research projects, namely ATESST2 [123] and MAENAD [79]. For a major part of the thesis, especially for the first two objectives, a two-step approach is used. In the first step, one or more systems (which are example cases for either timed automata or AUTOSAR) are modeled using EAST-ADL. In the second step, the results are validated and refined (if required) by modeling a representative case study for EAST-ADL using timed automata or AUTOSAR. The main focus has been on modeling formalisms, i.e., timed automata, state machine and discrete event formalisms. The tools are only used as a support for the work in order to achieve results independent of technology and tools. The work in the thesis can be divided into five parts and the methodology followed for each is described below.

1. The first part is an investigation of a relationship between the EAST-ADL timing model extension and timed automata. A case study of an *Emergency Braking System* (EBS) [85] was modeled in EAST-ADL using its UML profile implementation in PapyrusUML [95]. The EBS case study covers different aspects like timing constraints and logical behavior. An initial conceptual mapping between timed automata and EAST-ADL was derived using the case study. This mapping scheme is later used to model a brake-by-wire system with timed automata to validate and refine the mapping scheme. The work corresponds to arrow 1 in Fig. 3.2 and is presented in the appended papers A and B.

2. In the second part, a study of the EAST-ADL behavior description annex was carried out by modeling a BBW and an EBS system in MetaEdit+ [83], a domain-specific language tool. This was followed by utilization of the experiences and results from the first part of the thesis to develop an algorithm
for transforming models specified using the behavior description annex to timed automata. The work is presented in Paper C.

3. The purpose of the third part of the thesis is twofold. Firstly, a scenario of software relocation on a node failure is modeled using timed automata, followed by modeling the same scenario with EAST-ADL. This served the purpose of evaluating the extent of the support for dynamic configuration mechanisms provided by EAST-ADL and its integration with formal methods. Secondly, the validation of the transformation developed in the second part is carried out by applying it to the dynamic configuration scenario. The work is presented in Chapter 6.

4. The fourth part is concerned with the refinement of EAST-ADL models to AUTOSAR. In order to derive a relationship between EAST-ADL and AUTOSAR, two example case studies of a position control system and a fuel control system (providing coverage of various AUTOSAR artifacts) are first modeled in EAST-ADL. In the second step, the relationship is validated by modeling a brake-by-wire system in SystemDesk [36], a tool for AUTOSAR architecture specification. The work corresponds to arrow 2 in Fig. 3.2 and is presented in the appended paper D.

5. The fifth part of the thesis is related to the relationship between different tools supporting discrete event formalisms. For this work, efforts are made to model existing Stateflow models in SimEvents and vice versa. The results are then compared to derive the relationship between different formalisms and EAST-ADL. The work mainly corresponds to arrow 3 in Fig. 3.2 and is presented in Chapter 7.

### 3.4.1 Case Studies: Motivation and Usage

During the thesis several systems have been considered either directly or indirectly with varying purpose and details. These case study differ in their coverage such as AUTOSAR and timed automata. The following is an account of the case studies which are directly related.

1. The *Brake-by-Wire* (BBW) system is a representative industrial case used in several projects related to EAST-ADL. The main features are a structural model, requirements and timing constraints at multiple abstraction levels in EAST-ADL and a behavior model in Simulink. As part of the thesis, the BBW system is modeled in SystemDesk (a tool for specifying AUTOSAR software architecture), PapyrusUML (a UML tool with EAST-ADL profile), MetaEdit+ (a domain-specific language tool with an implementation of EAST-ADL meta-model) and UPPAAL (a timed automata tool). The modeling in SystemDesk and UPPAAL are carried out for validation of the results related to the relationship between AUTOSAR and timed automata.
respectively. The work with PapyrusUML was limited to the updating of the model with new EAST-ADL profiles when needed.

2. The Emergency Braking System (EBS) is originally presented in [85]. This case features automotive specific functions like radar and adaptive cruise controller with constraints related to timing, execution and network communications. In addition, the case also demonstrates refinements of an abstract model. For example, the abstract model of radar only covers execution behavior whereas its refined model includes additional logical behavior. In the thesis, the EBS case is modeled in PapyrusUML to explore the relationship between EAST-ADL and timed automata. The work also contributed in the development of the temporal constraint of EAST-ADL’s behavior description annex.

3. The Position Control System is an example case provided by dSpace with SystemDesk and TargetLink (a tool for model-based code generation). The case features basic software architecture and behavior, i.e., a PID controller modeled in TargetLink. In this thesis the case is used for a) understanding of AUTOSAR and how model-based development including analysis and code-generation is used in the context of AUTOSAR and b) to derive a relationship between EAST-ADL and AUTOSAR by modeling the system in PapyrusUML.

4. The Fuel Control System is also an example case provided by dSpace with SystemDesk and TargetLink. In comparison with the position control system, this case features additional behavior like modes, mode switches and related execution behavior of AUTOSAR software components.

5. A Dynamic Configuration Scenario which is a simplified case study based on the modeling and simulation work carried out in the earlier work discussed in Chapter 2. In the earlier work, several dynamic configuration scenarios related to load balancing and controlling of an application’s quality of service were modeled and simulated using Stateflow, SimEvents and TrueTime [92]. The purpose was to understand reconfiguration mechanisms, provide feedback during the development of the DySCAS middleware and demonstrate its capabilities. In this thesis the scenario is modeled with EAST-ADL and timed automata using MetaEdit+ and UPPAAL respectively. The purpose is to explore the extent to which EAST-ADL can support dynamic configuration mechanisms as well as further validation of the integration between EAST-ADL and timed automata developed in this thesis.

6. Automatic Drivetrain is an example case from Mathworks demonstrating the usage of Stateflow for automotive applications. An interesting feature of this case is that it includes different modeling formalisms. The shift logic of the drivetrain system comprises of modes, system states as well as different mathematical calculations. These characteristics are common for
both static and dynamically configurable systems. Furthermore, the EAST-ADL’s behavior description annex provides support for both computation and state machine behavior which makes the case highly relevant. In this thesis the case is remodeled in SimEvents to find a suitable relationship between Stateflow and SimEvents as well as to relate Stateflow and EAST-ADL with each other.

3.5 Summary of the Results

The thesis has the following five outcomes:

1. The first is a mapping scheme for transforming EAST-ADL models to timed automata. It provides formal interpretation for the timing model part of EAST-ADL in the form of timed automata. The work is presented in the appended papers A and B. Paper A presents an initial effort in transforming EAST-ADL timing models to UPPAAL and an investigation of automated transformation from an EAST-ADL model to UPPAAL XML based on Eclipse Modeling Framework (EMF) [39], an MDE technology. Paper B presents a template-based mapping of EAST-ADL timing constraints and timed automata. The template-based scheme is derived based on the experiences from Paper A.

2. The second outcome is a bridge between the behavior description annex of EAST-ADL and timed automata. In this work, an algorithm to transform Temporal Constraints (a part of the behavior description extension) to UPPAAL is presented with an automation prototype using MetaEdit+, a tool for domain-specific modeling with support for model transformation through its reporting language called MERL [84]. The work is presented in Paper C.

3. The third finding is a mapping scheme between EAST-ADL and AUTOSAR software architecture. The results are presented in Paper D. The resulting mapping scheme provides a basis for automated architecture refinements and synthesis. The mapping scheme has two parts related to structure and behavior respectively. A mapping between AUTOSAR and EAST-ADL has already been described in [30]. The new contributions of this thesis are mappings related to the timing model of EAST-ADL and identification of a few issues related to the refinement of EAST-ADL models to AUTOSAR. The result also provides an update of the mappings presented in [30] with the latest version of EAST-ADL.

4. The fourth outcome is an evaluation of the extent to which EAST-ADL and its integration with formal methods can support modeling and analysis of dynamic configuration mechanisms. The work is presented in Chapter 6 and will be published in the form of a research article.
5. The fifth outcome is a comparison of Stateflow and SimEvents in the context of EAST-ADL behavior description annex. The work is presented in Chapter 7 and will be published in the form of a research article.

3.6 Discussion

This chapter has provided an overview of the thesis, the main motivations of which are a) the need for integration of different models, formalisms and tools into a seamless integrated environment and, b) the evaluation of existing technologies for their potential to support advanced and future functionalities. Furthermore, the work addresses both horizontal and vertical consistencies. While the mapping of EAST-ADL configurations to AUTOSAR parameters contributes to vertical consistency, the transformation from EAST-ADL to timed automata contributes to horizontal consistency.
Chapter 4

State of the Art and Related Work

This chapter presents an overview of the state of the art related to the thesis. The overview of the state of the art starts from Section 4.1 and ends at Section 4.4. It can be divided into three different parts covering architectural description languages applicable for automotive systems, tools supporting heterogeneous modeling and simulation and, model checking and timed automata for verification of real-time systems. The related work is presented in Sections 4.5, 4.6 and 4.7. The chapter is concluded with a brief discussion in Section 4.8.

4.1 Architecture Description Languages

Architecture description languages (ADL) (defined in Section 1.1.4) provide a means for documentation and communication of the architecture of a system under consideration in a structured way [62]. An ADL is also an approach for model integration [22]. Some languages are developed explicitly as ADLs while others are implicit. The template-based architecture description of the AUTOSAR standard is an example of an implicit ADL. The following sections describe a few selected ADLs. EAST-ADL being targeted ADL of the thesis is covered in detail as compared to other ADLs.

4.1.1 EAST-ADL

EAST-ADL (Electronics Architecture and Software Technology - Architecture Description Language) has evolved from several European projects since 2001. Fig. 4.1\textsuperscript{1} shows an overview of EAST-ADL. Also notice the similarities between Figures 4.1 and 3.2. The following are the four main abstraction levels specified by EAST-ADL [119]:

\footnote{\textsuperscript{1}With permission from the EAST-ADL association [38]}
**Vehicle level** for specification of product features and their relations like mandatory, optional and alternative. For example, while ‘simple braking’ is a mandatory feature for every vehicle ‘advanced braking’ can be an optional one.

**Analysis level** for specification of system functionalities and their connections realizing different features. For example, a braking feature is realized by functions including brake pedal sensing, torque calculation, actuation, etc.

**Design level** is related to the logical design of a system which refines an analysis level model with specification of additional functional components, hardware topology and the allocation of functional components on different hardware components. For example, a sensing function can be refined to components related to transfer function of a sensing device, device driver and a middleware component for interfacing applications and the device driver component.

**Implementation level** related to detailed implementation design at the AUTOSAR level. The design at this level includes but is not limited to software components, their behavior and assignment of software components to hardware nodes and operating system tasks.
As shown in Fig. 4.1, the approach followed by EAST-ADL is modular with a core package consisting of structural attributes and extensions for specifying requirements, timing and behavior constraints, dependability and variability of different components along with verification and validation of a system.

The core structural package whose subset in its simplified form is shown in Fig. 4.2 is used to specify components and their parts like ports and connectors at different abstraction levels. The components at one abstraction level are related with another abstraction level by ‘realization’ links. For example, consider a FunctionalDevice (a specialization of AnalysisFunctionType not shown in the figure) which serves as an interface between the plant and the embedded part. A functional device at the analysis level can be realized by one or more DesignFunctionTypes at the design level of abstraction. The DesignFunctionTypes can be related to device driver, middleware component, etc.

The extensions serves the following different purposes depending on their type:

- The Requirements model [119, 122] is derived from SysML. It is used to specify different requirements and their relation with each other using ‘derived’ relation or with other EAST-ADL artifacts like a DesignFunctionType using ‘satisfy’ relation.
• The Dependability model [119, 120] is used for error modeling describing faults, failures, safety level assignment, etc. The readers are referred to [21] for an overview of the support for an integrated safety and architecture modeling leveraged by EAST-ADL.

• The Timing model is used to specify constraints on timing related execution such as execution rate of a component, end-to-end delays, etc. The readers are referred to [117] for basic concepts related to the timing extension.

• The Variability model [119, 121] can be used to specify product line variations.

• The Verification and Validation (V&V) [119, 122] extension can be used to specify procedures, expected and actual outcomes to validate or verify a system. EAST-ADL provide support for relating requirements as well as core structural artifacts with V&V artifacts.

• The Behavior model [119] can be used to specify behavior in terms of mode, type of trigger, i.e., event-based or time-based, representation type such as Simulink, SCADE, etc.

• The Behavior Description Annex [126] for defining constraints on behavior specified by the behavior model.

Figure 4.3: Events and event chains in EAST-ADL [119]

**Timing Model**

The timing model of EAST-ADL is derived from TADL (Timing Augmented Description Language) [129]. It can be used to specify the timing constraints on the execution of functions including end-to-end delays and precedence between
different functions. As shown in Fig. 4.3 the timing extension is based on the concepts of events and event chains. *EventFunction*, *EventFunctionFlowPort* and *EventFunctionClientServerPort* are the three event kinds referring to the triggering of a function by some sort of dispatcher, arrival of data at a port and service requested (or received) by a client-server port respectively.

As the name indicates, an *event chain* is used to specify chain of events. An initial event is termed as a *stimulus* and a final event is termed as *response*. An event chain can have multiple stimuli and responses. It can further be refined into smaller event chains called *strands* (parallel chains) or segments (sequenced).

![Figure 4.4: EAST-ADL timing constraints](image)

With the timing extension it is possible to specify constraints (shown in Fig. 4.4) on the events and event chains. The periodicity of an event occurrence is an example of possible constraints on the events. For additional information, the readers are referred to [119].

**Behavior Model**

EAST-ADL makes a distinction between a specified behavior and function behavior. While the function behavior is the implemented behavior, a behavior specification is the expected behavior or its constraints. In other words, the behavior specifications corresponds to what the behavior should be and the function behavior is related to the implementation of function behavior in different forms like a Simulink function or a programming code.
Currently, the behavior model and the behavior description annex are two different packages in EAST-ADL. The behavior model (parts of which are shown in Fig. 4.2 with a prefix ‘Behavior::’) enables the specification of behavior realization means of a function in terms of triggering of a function, operational modes and links to the actual behavioral models. The triggering is defined by ‘FunctionTrigger’ which mainly specified if the triggering is periodic or event driven. The links to the actual behavior is specified through the property ‘representation’ of ‘FunctionBehavior’.

**Behavior Description Annex**

![Diagram](image)

Figure 4.5: Behavior Constraints and its relation with other EAST-ADL artifacts [126]

The recently developed behavior description annex [126] (BDA) shown in Fig. 4.5 is a framework for formalization of behavior concerns. It can be used for several purposes such as refinement of the specification of a feature (shown by the relation targetedVehicleFeature in Fig. 4.5) or a component’s internal behavior (shown by the relation constrainedFunctionBehavior in Fig. 4.5) or end-to-end behavior of a group of components, etc. The BDA also provides a way to relate artifacts from different extensions for the traceability of behavior specifications. This is supported by the explicit syntactical association of the artifacts from the behavior extension and other extensions. For example, in a state machine type representation, if the behavior of a component has two states, then one state can correspond to an operational mode defined using the behavior annex and the other can correspond to an error state defined using the dependability extension.

The behavior description annex is classified into three categories [126]:

- **Attribute quantification constraint** related to the declarations of value attributes and the related acausal quantifications (e.g., $U=I^aR$).
4.1. Architecture Description Languages

- **Temporal constraint** related to the declarations of behavior constraints where the history of behaviors on a time-line is taken into consideration.

- **Computation constraint** related to the declarations of cause-effect dependencies of data in terms of logical transformations (for data assignments) and logical paths.

The behavior description annex is aligned to hybrid system models as well as timed automata formalism. In temporal constraint shown in Fig. 4.6, a state can refer to a generic location or a specific operational mode, error state and hazard. The two kind of possible state invariants are quantification (related to some parameters) and time (related to clocks). The same applies for the transition guards. Additional guards for synchronization in terms of event occurrence can also be specified. The events are of two types namely timing and logical. The timing events are defined by the timing constraints and the logical events refer to an event where one or more parametric conditions become true.

The other constraint types are out of scope of the presented work, therefore, the readers are referred to [126] for further details.

### 4.1.2 AADL

AADL (Architecture Analysis and Design Language) [1] was originally developed for avionics systems development, and is based on its predecessor MetaH. It shares many properties with EAST-ADL but has a smaller scope. The focus of AADL lies on the detailed architecture specification and verification. The language
can either be adopted as part of a top-down development approach, supporting
detailed architecture design and integration analysis, or for reverse engineering
efforts in modeling and analyzing (changes to) legacy systems. The language
specifications do not explicitly specify more than a single level of abstraction or an
implied development approach. Similarly to EAST-ADL, it also has core language
constructs and the extensions namely error modeling for dependability aspects
(e.g., reliability and availability), meta-model for XMI/XML annex for supporting
additional tools, notation for graphical modeling, behavior modeling, ARINC 653
to support the standard for aircraft industry and UML2.0 profile to support the
use of UML.

The most common tool used for modeling with AADL is OSATE (An Extensible
Open Source AADL Tool Environment) [127]. It is possible to perform resource
and end-to-end latency flow analyses, semantic analysis of modal systems as well
as security level checking directly within the tool environment. This is exemplified
by the AADL consortium with different case studies [127]. AADL has also been
incorporated in the latest MARTE standard [91]. AADL also has a few drawbacks,
such as complex component compositions and property ambiguity [48].

In comparison with EAST-ADL, AADL supports a subset of architectural
specifications. In some cases extensions to AADL has been proposed. SLIM
(System-Level Integrated Modeling) language [124] is an example of such extension
to enable timed modeling of behavior, linear time-dependent dynamics, etc. EAST-
ADL being developed based on the state of the art technologies incorporates many
of the features which are missing in AADL.

4.1.3 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [6] is a software development
standard for dealing with the ever growing complexity of automotive embedded
systems. It provides a basis for modular software architecture with standardized
interfaces and specifications of a run-time environment. It is also possible to use
commercial-off-the-shelf components and the components obtained from different
suppliers, as long as they provide AUTOSAR compliant interfaces.

The result of the AUTOSAR effort is a framework and methodology [28] for
standardized automotive software and hardware. A six layered software architecture
is a part of the AUTOSAR framework. The layers are Application, Run Time
Environment (RTE) providing a middleware functionality, Service, ECU (Electronic
Control Unit) Abstraction, Complex Drivers, and Micro-controller Abstraction.
While the ECU abstraction is the lowest layer, the Application is the highest one.
These six layers together comprise of more than 49 basic software modules ranging
from modules for communication, drivers for different devices to services for memory
and processor.

An AUTOSAR software component is also called an Atomic Software Com-
ponents as it cannot be distributed over several ECUs. AUTOSAR provides
separation of application and infrastructure. While the former is related to
providing software functionalities such as control algorithms, etc. the latter is used for different services related to an ECU. In addition a concept of a Virtual Functional Bus (VFB) is also introduced to abstract the interconnections between different software components. Furthermore, AUTOSAR supports both client-server and sender-receiver kind of communication between its components.

As the work presented in this thesis is focused on behavior, it is to be noted that the internal behavior of an atomic software component describes aspects related to the Run-Time Environment (RTE). An RTE provides communication abstractions to the atomic software components [6]. Runnable entities (smallest code fragment provided by a component) capable of running under different modes and RTE events (e.g., reception of remote invocation, timeout, etc.) are examples of the behavioral aspects.

For the description of architecture and its components standard templates are provided. These templates correspond to the ADL part of AUTOSAR providing a means to specify and relate different architectural entities.

AUTOSAR and its ADL are evolving with time. The latest release, i.e., version 4.0, provides additional support for dynamic scheduling and introduction of multi-core architectural concepts. The current evolution includes extensions to address non-functional aspects such as safety (partial alignment to ISO 26262 safety standard [61]) and timing (findings from the TIMMO project - TADL [129]).

For more information, the readers are referred to the specifications available on the AUTOSAR website [6].

4.1.4 MARTE

MARTE (Modeling and Analysis of Real-Time and Embedded System) [91] is a standard from OMG (Object Management Group)\(^2\). It can be considered as an architecture description language providing support for describing real-time systems. The standard has three major parts. The first part provides the foundations enabling specification of functional properties like execution and logical behavior and run-time context in which a particular behavior can occur. The first part also provides support for specifying non-functional properties, modeling of time and resources.

The second part of MARTE is related to the design models which correspond to the structure part of an embedded system. It mainly includes structural models (i.e., components, ports, connectors, etc.), application models (i.e., quantitative features such as period and deadline as well as qualitative features related to behavior, concurrency and communication), and detailed resource (software and hardware) modeling support.

The third part of MARTE is used for specifying analysis models. The analysis models can be generic as well as specific for schedulability and performance (sensitivity, scalability or capacity, etc.) analysis.

\(^2\)http://www.omg.org/
In addition to above, the standard also provides guidelines for its usage with other ADLs like AADL and EAST-ADL.

4.2 Heterogeneous Modeling and Simulation

Embedded systems are heterogeneous where each component or subcomponent can have different characteristics and hence suitably modeled with different models of computation. This also applies to the methods and protocols for inter-component communications. Due to these differences, the analysis of such systems is non-trivial [40]. The following are a couple of environments supporting modeling and simulation of heterogeneous systems.

4.2.1 MATLAB / Simulink

MATLAB is a programming tool from Mathworks [139] which along with its graphical extension Simulink is extensively used in various industries. MATLAB can be used for developing different kind of algorithms related to engineering disciplines like control systems and signal processing as well as for data analysis and visualization. Simulink is used for multi-domain simulation and model-based development of dynamic systems. Together with the ad-on products like Stateflow, MATLAB/Simulink can be used to model various kinds of models of computations related to continuous time, discrete time and event based simulation. From a model-based development perspective, it is possible to automatically generate code from Simulink models after analysis, relate models to requirements, perform verification and validation for a given set of requirements, etc.

SimEvents

SimEvents is a toolbox for Simulink for simulating discrete event systems. Entities and events are the two basic concept used in SimEvents. While the former refers to “discrete items of interest” [58] such as packets and frames in a communication network, the latter refers to a discrete incident such as change of state or occurrence of other events like a function call [58]. According to an earlier evaluation [52], SimEvents provide most of the desirable features for discrete event system simulations.

The different blocks included in the library of version 2.4 of SimEvents include support for generating entities both periodically and aperiodically, association of data with entities and manipulation usable for implementing algorithms and decisions, queues of various lengths which can be utilized to model structure such as port dynamics. One of the major features is the possibility of assigning time to different entities which can utilized in several ways such as scheduling of different services, discarding old values depending on the implemented policy, etc.

A few applications where SimEvents have been used for simulation include queues for manufacturing systems [87], anti-lock brake system (ABS) with CAN
(Controller Area Network) [24] and modeling of a nuclear facility [75]. SimEvents has also been used to simulate real-time systems including the Ethernet network, operating system and memory management [5].

### 4.2.2 PtolemyII

PtolemyII [40] is an open-source tool supporting the simulation of multiple models of computation resulting from the efforts in the Ptolemy project [102]. It takes a hierarchical actor-oriented approach where the semantics of the models are determined by a part of the model called the director. A director can be assigned for each level of hierarchy. The approach emphasizes concurrency and communication, and well defined interactions between different MoCs. The tool has a graphical interface called Vergil.

### 4.2.3 TrueTime

TrueTime [92] is a MATLAB/Simulink based simulator for real-time control systems. It targets embedded systems software, real time operating systems, distributed control and networks. It is also possible to predict performance through simulation. TrueTime provides abstractions for a basic RTOS (Real time operating system) with messaging services, and simulation of networks and network interfaces. A very basic interface for simulation of controlled power consumption is also available. Concepts like higher level communication protocols, middleware, automatic power management, etc, are out of the scope of TrueTime and need to be explicitly modeled by the modeler or added to the toolbox library.

### 4.2.4 Metropolis

Metropolis and its successor Metro-II [31] is a design environment based on SystemC for designing heterogeneous systems. A platform-based design methodology is followed when using this tool. This methodology defines a “meet in the middle process” where at each level of abstraction, the available details about the implementation are sufficient to simulate a system. The generic steps followed in the methodology are functional decomposition, behavior adaptation, communication refinement, mapping and optimization, etc.

A few of the formalisms supported by the tools are Mealy and Moore machine, synchronous data-flow, etc. The major entities of Metro-II are process (implementing a MoC) and media (communication functions and variables), interfaces, properties, constraints (temporal and quantitative), firing conditions.

Overall, Metropolis is a co-simulation environment where both hardware and software can be co-simulated with well-defined interfaces. Compared to its predecessor, Metro-II provides the additional features like the ability for importing pre-designed IPs, separation of cost from behavior during the design, and a structured design space exploration.”
4.3 Timed Automata and Model Checking

Model checking is a powerful method to verify a design of a system. In contrast to a simulator, a model-checker can traverse through all the possible system states. Model checking is also used for verifying system specifications. There exist a lot of model checking techniques applied on different variants of automata and implemented on different tools for verification of real-time systems. The following discussion focuses on timed automata.

Timed-automata (TA) [11] constitutes automata augmented with time semantics to enable formal analysis of real-time systems. The timed semantics are supported by the notion of clocks and constraints applied on them. For example, to specify the maximum time duration for which a location (or state) can remain active, a clock invariant is used. Similarly, it is also possible to specify time conditions on transitions between two locations.

Often a set of timed automata is used in a networked form with a common set of clocks and actions. A special synchronization action denoted by an exclamation sign (!) or a question mark (?) is used for synchronization between different timed automata. A timed-automaton in a network is concurrent unless and until mechanisms like synchronization actions are applied. The readers are referred to [11] for a formal definition and semantics of a network of timed automata.

UPPAAL [8] is a model checking tool based on timed automata for modeling, validation and verification of real-time systems. The tool has three main parts: an editor, a simulator and a verifier, for modeling, fault detection (by examination, i.e., without exhaustive checking) and verification (covering exhaustive dynamic behavior) respectively. A system in UPPAAL is modeled as a network of timed automata. A subset of CTL (Computation Tree Logic) is used as the query language in UPPAAL for verification. In addition to the generic timed automata, UPPAAL uses the concepts of broadcast channels for synchronizing more than two automata. The concept of urgent and committed state is also introduced to force a transition as soon as it is enabled. Similar to other automata tools, the three kinds of properties which can be checked using UPPAAL are

1. **Reachability**, i.e., some condition can possibly be satisfied,
2. **Safety**, i.e., some condition will never occur and
3. **Liveness**, i.e., some condition will eventually become true.

UPPAAL uses the concept of *templates* for reusability and prototyping of system components. Each template can be instantiated multiple times with varying parameters. An instance of a template is called a *process*. The tool has been used in many industrial cases such as a gear box controller from Mecel AB [77] and audio protocol from Philips [10].
4.4 Architectures Supporting Dynamic Reconfigurations

Dynamic reconfigurations are one of the key enabling technologies for future vehicular systems [4]. Dynamic configurations is supported by both hardware technologies like FPGAs and software technologies like middleware. Reconfigurable hardware is out of scope of this work. The middleware developed for dynamic and adaptive systems include but are not limited to DynamicTAO [71], 2K [70], RUNES [108], OSGi [93], and DySCAS [37].

In addition there exist other technologies targeting heterogeneity such as the SHIFT language [33] through which dynamic systems can be modeled and simulated as network of hybrid automata.

4.5 Integration of ADLs with Analytical Formalisms

Analysis of models described by different architectural description languages using external tools has been a focus of many researchers. This is true especially for AADL. The COMPASS (Correctness, Modeling and Performance of Aerospace Systems) project [124] developed an extended AADL language called SLIM enabling modeling of systems with a combination of nominal and error behavior of components. The two models are related to each other by additional semantics related to fault injection. The EAST-ADL behavior description support has a broader coverage as compared to SLIM but it can also be utilized to follow the same strategy as that of SLIM. In [66] AADL is integrated with UPPAAL for formal verification of architectural specification to ensure completeness and consistency.

A number of efforts have been carried out to enable the analysis, verification and validation of system architecture design captured in EAST-ADL. [43] presented an effort to integrate the SPIN model checker for formal verification of EAST-ADL models. The automata addressed by SPIN are untimed and the SPIN transformation needs to be updated for the latest EAST-ADL release. The authors of [80] proposed the use of MARTE for complementing EAST-ADL to enable timing analysis.

A method for timed automata based analysis of EAST-ADL models is presented in [68] where timed automata are used as external models for behavioral modeling and analysis. In contrast this thesis focuses more on the native EAST-ADL artifacts for modeling and external models for analysis.

The timing constraint package used in this thesis is a subset of TADL (Timing Augmented Description Language) [129] developed by the TIMMO project consortium.

One of the interesting model-transformation approaches related to ADLs is the TASTE [97] method providing and ADL approach for system integration. It provides a domain specific graphical editor with an AADL generator. Among others, it provides support for schedulability analysis, system execution and
detection of possible errors like deadlock. The tool-chain is capable of integrating code generated from different environments like MATLAB and SCADE.

In addition to the above [86] presented a component-based framework for modeling real-time systems and integrated the framework with timed automata for analysis.

4.6 Modeling and Simulation of Dynamically Configurable Architectures

Several efforts have been carried out in terms of simulations of dynamic configurations. While, some efforts are focused on the simulation of dynamic reconfiguration through hardware architectures like FPGA, the others are focused on software technologies like middleware. The following discussion provides a brief overview of both types of efforts.

4.6.1 Middleware Simulations

A few efforts carried out for the simulation of different middleware technologies are as follows. WISDOM (Wsn mIddleware Service moDules simulatiOn platforM) [76] framework written in the Java language for simulation and verification of different middleware protocols in wireless sensor networks. The framework is developed and used for simulation and verification of services related to routing, target detection and tracking, etc. [142] presented a TOSSIM (TinyOS mote simulator) based simulation of the Agilla middleware [45]. Similar to WISDOM the target is also wireless sensor networks.

UCS (Ultra CORBA Simulator) [118] is one of the simulators for simulating CORBA (Common Object Requesting Broker Architecture) [29] middleware. The provided support in UCS includes simulation for both client and server sides, naming service, GIOP (General InterORB Protocol) and various other functionalities. The former two methods, i.e., the client-server and naming/trading mechanisms are also used in the DySCAS middleware.

4.6.2 Platform-Based Modeling and Simulations

Detailed hardware modeling and simulation techniques such as for cycle accurate model [46] are out of the scope of this thesis. However, for simulating embedded systems, some abstraction of hardware is always required. In other words, platform-based simulations are required. There exist a lot of examples of platform-based simulations. The following are a few examples related for this thesis.

In [141] simulated queue management system for networked control systems. Similarly, [5] demonstrates abstraction of applications, operating system for priority based scheduling, shared memory management, etc. The simulations are used for identification of constraints on shared resources and exploration of different algorithms for resource optimization.
Analysis of systems with shared resources and networked systems has also been carried out using TrueTime. Several applications of TrueTime based simulations are presented in [92] such as analysis of control performance with varying execution time of control tasks, varying scheduling schemes and introducing delays and losses in network communications.

4.7 Efforts for Integrating Discrete Event Formalisms and Tools

A lot efforts have been made to integrate different discrete event formalisms and tools. In [94], UPPAAL models are transformed to Stateflow in the context of a pacemaker development. The results can be utilized to extend the work presented in this thesis. Similarly, [112] presents a method to transform Stateflow models to Lustre. The objective was to transform a sequential language to a synchronous language. Integration of different modeling formalism been addressed in the form of interfacing different models of computations. There exist a lot of literature on such issues. A couple of them are the books from Gajski et. al. [46] and Jantsch [64]. Some methodologies like ForSyde [109] deal with the usage of different models of computations for embedded systems development.

In contrast to the above mentioned works which are mainly related to different tools, this thesis is centered around architecture description language. The main purpose is to find a way to make architecture models and the analysis models in different tools as consistent as possible.

4.8 Discussion

This chapter briefly presented a few of the state of the art technologies related to model-based development in general as well as architectures supporting dynamic reconfigurations. Various architecture description languages target different models of computation and formalisms. For example, EAST-ADL can support formalisms like timed automata and continuous-time in terms of behavior and component-based modeling for structural specifications. These formalisms can have a direct relationship with the formalisms supported by the tool environments for modeling, simulation and analysis of embedded systems. In some more challenging cases, a straightforward relation may not exist due to varying dialects mentioned in Section 3.1. Out of the three usages of model transformation mentioned in Section 1.2.3, the thesis is mainly focused on analysis and simulation. However, the work also contributed towards tool integration. The next chapter presents results from the work on integrating EAST-ADL with timed automata and AUTOSAR.
Chapter 5

EAST-ADL Integration with Timed Automata and its Connection with AUTOSAR

This chapter summarizes the results related to research questions 1 and 2 described in Section 3.3 with an exception of question 1c which is addressed in the next chapter. With EAST-ADL, timed automata, and AUTOSAR as the targeted technologies, the focus is on model transformations for analysis and synthesis\(^1\).

Model transformation for analysis is in the form of integration between EAST-ADL models defined using its Timing Model and Behavior Description Annex to timed automata. The work is presented in the appended papers A, B and C and summarized in Section 5.1. One of the major assumptions made for the horizontal integration (with timed automata) was that only one EAST-ADL design function type which corresponds to *Functional Design Architecture* (FDA)\(^2\) can contain prototypes of other functions. Without this assumption, a mechanism to flatten the architecture will be required as timed automata does not support hierarchical modeling. Refer to the appended Paper C for further details.

The transformation of EAST-ADL configurations to AUTOSAR parameters is a model transformation for synthesis. The relationship between EAST-ADL and AUTOSAR is presented in Section 5.3. The chapter is concluded with a summarizing discussion in Section 5.4.

5.1 EAST-ADL Timing Model and Timed Automata Integration

The integration of timing model and timed automata is supported by two case studies. The first case study is an Emergency Braking System (EBS) model

\(^1\)Refer to 1.2.3 and [13, 82] for the types of usage of model transformations

\(^2\)An FDA refers to the software architecture part at the design level of abstraction
Chapter 5: EAST-ADL Integration with Timed Automata and its Connection with AUTOSAR

presented in [85] where the authors show how large scale real-time systems can be analyzed using timed automata. The authors of [85] first developed a primitive EBS model and verified it for different properties such as execution time of functions. This was followed by refinement of the model and its verification for the same properties. The second case study is a Brake-by-Wire (BBW) system. The system is a representative case for EAST-ADL and has been used in several EAST-ADL related projects.

As mentioned earlier in Section 3.4, a two-step approach is adopted for the work. The EBS model was first transformed to EAST-ADL for deriving and reasoning about a mapping scheme. This was followed by transforming the BBW model to timed automata for further validation.

Paper A presents an initial effort where an algorithm was developed along with its prototype implementation. The implementation is based on the Eclipse Modeling Framework [39] with a complete UPPAAL meta-model and a lite version of the EAST-ADL meta-model. Based on the experiences from the work presented in Paper A, timed automata templates were developed along with their extension guidelines presented in Paper B. The following sub-sections present the mappings related to the EAST-ADL function types and a few selected timing constraints. For details the readers are referred to the appended papers A and B.

A template-based solution is proposed where a timed automata template for each timing constraint (delay, precedence, etc.) and execution (periodic or aperiodic) semantics of a function type is developed. The templates are divided into three types. The first type models EAST-ADL function types (specified by the EAST-ADL core structure) and their execution behavior (specified by the EAST-ADL’s timing model). In particular the execution behavior corresponds to the execution time constraint shown in Fig. 4.4. The second type of template is called ‘Rate Transition’ which is proposed to handle the communication between EAST-ADL functions having different execution frequencies. The third part is related to the timing constraints specifying bounds on individual events and event chains. This in turn corresponds to all the constraints shown in Fig. 4.4 apart from the execution time constraint. The three parts are summarized in the following text.

5.1.1 Functional Components

The execution behavior of a functional component (such as a design function type or its prototype) in EAST-ADL is mapped to an automaton with three or two locations corresponding to time or event triggering respectively as shown in Fig. 5.1. The triggering is defined using the FunctionTrigger (Fig. 4.2). For event based execution a port is specified with the FunctionTrigger. For a time-based execution, an event (i.e., ‘EventFunction’ in Fig. 4.3) is specified followed by the specification of the execution time (‘ExecutionTimeConstraint’ in Fig. 4.4) for the event.

The input? and output! synchronization actions in Fig. 5.1 correspond to data
5.1. EAST-ADL Timing Model and Timed Automata Integration

The wire system in EAST-ADL to timed-automata. The same approach is used for the proposed mapping scheme and its validation. This mapping consists of templates for each function and timing constraint type. The templates for the timing constraints act as monitors indicating if a constraint is met or not. In addition to the description of the semantics of the mapped EAST-ADL artifacts, template implementations in UPPAAL are also presented for illustration.

Event. In terms of timed automata, an event can be modeled as a synchronization action. For example, the synchronization action `output!` for the transition to the final from the execute state shown in (b) can be considered as an event corresponding to an EventFunctionFlowPort referring to a port with direction `out` or EventFunctionClientServerPort with kind of either `sentRequest` or `sentResponse`.

![Function templates](image)

Figure 5.1: Function templates

arrival and departure from a non-triggering\(^3\) port respectively.

A variation in the template exists for the cases where a function has only an input(s) or an output(s). This is illustrated for the function pBTC of the brake-by-wire model in Fig. B.18 of the appended paper B. The decision of including or excluding the input and output synchronizations depends on the existence of the ports and their flow (in or out) within a function under considerations.

The execution time bounds, i.e., minexec- and maxexecTime are the properties of the execution time constraint (shown in Fig. 4.4 in Paper B) associated with the function under consideration.

### 5.1.2 Rate Transition

In model checking tools like UPPAAL, a deadlock occurs if a synchronization action associated with an enabled transition cannot be executed. In contrast, an EAST-ADL system is deadlock free due to the single-sized over-writable buffer semantics of a function port. This means that if a data arrives at a function port, the old value is overwritten. Furthermore, a function type does not wait for sending or receiving data on its port. To handle this issue two different rate-transition templates shown in Fig. 5.2 are introduced which provide a basis for creating intermediary processes between two functions running at different frequencies. The input to the rate transition blocks correspond to the output of the sending function and vice versa.

For the fast to slow rate transition, it is assumed that the frequency of the fast running function is an integer multiple of the the slow running function. The rate difference is defined by the variable ‘difference’ as an input parameter to the template.

\(^3\)i.e., a port which is not responsible for triggering a function
Chapter 5: EAST-ADL Integration with Timed Automata and its Connection with AUTOSAR

5.1.3 Timing Constraints

Each timing constraint type defined in the Timing Model package of EAST-ADL [119] is translated to an UPPAAL template and in turn a timed automaton. Each template comprises of a minimum of three locations corresponding to the initial state, a failure state (i.e., constraint not satisfied) and a success state. The input events (synchronization action in timed automata) of the instantiated templates can correspond to a single or a combination of multiple EAST-ADL events. Different EAST-ADL events are shown in Fig. 4.3. The constraints can be applied on individual EAST-ADL events or event chains. Execution time constraint applied on the data arrival/departure or service receive/sent from a port specified by ‘EventFunctionFlowPort’ in Fig. 4.3 is an example of a constraint applied on individual events. Delay and precedence constraints are examples of constraints applied on event chains.

Figure 5.2: Rate transition templates

a) Fast to slow rate transition b) Slow to fast rate transition

5.1.3 Timing Constraints

Each timing constraint type defined in the Timing Model package of EAST-ADL [119] is translated to an UPPAAL template and in turn a timed automaton. Each template comprises of a minimum of three locations corresponding to the initial state, a failure state (i.e., constraint not satisfied) and a success state. The input events (synchronization action in timed automata) of the instantiated templates can correspond to a single or a combination of multiple EAST-ADL events. Different EAST-ADL events are shown in Fig. 4.3. The constraints can be applied on individual EAST-ADL events or event chains. Execution time constraint applied on the data arrival/departure or service receive/sent from a port specified by ‘EventFunctionFlowPort’ in Fig. 4.3 is an example of a constraint applied on individual events. Delay and precedence constraints are examples of constraints applied on event chains.

Figure 5.3: Periodic event constraint template
5.1. EAST-ADL Timing Model and Timed Automata Integration

**Periodic event constraint:** A periodic event constraint is used to specify constraints on the periodicity of an event. An UPPAAL template for a periodic event constraint is shown in Fig. 5. The three applied parameters (also shown in Fig. 3) in this template are period (P), jitter (J) and the minimum arrival time of the event. The synchronization action "event?" can refer to any event whose periodicity is required to be constrained.

*Fig. 5. Periodic event constraint template*

**Reaction constraint:** A reaction constraint specifies a bound between the occurrences of stimuli and responses of an event chain. According to [3] there exist five possible specification combinations ({upper, lower}, {upper, lower, jitter}, {upper}, {lower}, {nominal, jitter}) for a delay constraint. The presented work considers only one combination i.e. {upper} which corresponds to the maximum time allowed.

*Fig. 6. Reaction constraint template*

In the reaction constraint template (Fig. 6a) the clock is reset when a stimulus event occurs. As soon as the response event occurs the automata transits to Fail or Success state depending on the elapsed time i.e. the LocalClock value. The template considers only one stimulus and one response. It can be extended for multiple stimuli and responses by adding additional states and parallel transitions. For example, in case of two stimuli, two states between the Init and Wait states e.g. s1 and s2 can be added where the transition from Init to s1 corresponds to the first stimulus occurrence, s1 to Wait corresponding to the second stimulus and vice versa. This is illustrated in Fig. 6b.

**Precedence constraint:** A precedence constraint specifies the constraint on the order of execution of events. A template for two events is shown in Fig. 7 where input2 is constrained to occur after input1. In order to extend this template for more events...

**5.1.4 Verification**

For verification, the consistency checking essentially becomes a safety property (defined in Section 4.3), where it has to be ensured that for a given timing constraint template instantiation, the corresponding automaton never reaches the Fail state. This is specified as $\forall x (\neg xx.Fail)$ in the UPPAAL query language where ‘xx’ is the instantiated template for a timing constraint.

**5.1.5 Mapping Summary and Discussion**

The summary of the mapping is given in the table shown in Fig. B.12 in the appended paper B. The mappings also abstract the execution schedule on a given hardware platform by synchronization actions in timed automata. This in turn requires that the sum of maximum execution time of all the functions having the same periodicity and allocated to same processing units is less then or equal to their period. This ensures that functions when refined to an implementation will be schedulable. For other cases, it is recommended to perform a schedulability analysis for each set of functions allocated on a single processor. This can be done by tools like Times [2].

The above mapping concept has been validated experimentally by changing configuration specifications and performing schedulability analysis for each set of functions allocated on a single processor using the Times tool. In the schedulability
Chapter 5: EAST-ADL Integration with Timed Automata and its Connection with AUTOSAR

analysis, the actual computing time was assumed to be equal to the maximum execution time. Similarly the deadline was assumed to be equivalent to the period.

The fast to slow rate-transition template is applicable for cases where the frequency of a fast running function is an integer multiple of that of the slow running function. To overcome this limitation the use of data and value guards can be considered. This requires further investigation and is left for future.

The work on mapping the EAST-ADL timing model to timed automata was carried out without using the EAST-ADL behavior description annex (BDA). In a wider scope, the integration of EAST-ADL timing model and timed automata can be facilitated by the temporal constraint part of the BDA. With BDA, it is not only possible to model the timed automata templates within EAST-ADL but also to augment them with additional logics. The next section summarizes the efforts to transform temporal constraints to timed automata.

5.2 Behavior Description Annex Integration with Timed Automata

The thesis focuses on the temporal constraint part of the EAST-ADL behavior description annex (as introduced in Section 4.1.1). In contrast to the timing model, the temporal constraints have a timed automata based semantics. In this part of the thesis, a transformation algorithm was developed followed by its verification by its prototype implementation using MERL [84].

5.2.1 Transformation Algorithm

As shown in Fig. 4.5, all BehaviorConstraintTypes can have one or more BehaviorConstraintPrototypes. Furthermore, a BehaviorConstraintType can be instantiated many times in the same or different BehaviorConstraintTypes in the form of a BehaviorConstraintPrototype. Different behavior prototypes are composed together using binding parameters and events. The binding parameters may or may not be associated with a port of an EAST-ADL function.

A major assumption made for the transformation algorithm is that only one BehaviorConstraintType (referred to as the main behavior constraint type in this section) is allowed to have prototypes of other BehaviorConstraintTypes and that it is not allowed to have an instantiation of its own prototype. The behavior constraint type with one or more prototypes is the starting point of the algorithm. In addition all the binding parameters are assumed to be associated with a port.

The algorithm to transform temporal constraints to EAST-ADL is based on the conceptual mapping shown in Fig. 5.5 and described as Algorithm 1 in Paper C. The algorithm in its simplified form is as follows:

1. For every behavior constraint type except the main one define a timed automata template as follows:
5.2. Behavior Description Annex Integration with Timed Automata

<table>
<thead>
<tr>
<th>EAST-ADL</th>
<th>Timed Automata (UPPAAL)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemporalConstraint</td>
<td>Process declaration</td>
<td>The temporal constraint is like a container where all the states, transitions and events are declared.</td>
</tr>
<tr>
<td>BehaviorConstraintPrototype</td>
<td>Process instantiation</td>
<td>It is an instantiation of a BehaviorConstraintType in another one.</td>
</tr>
<tr>
<td>BehaviorConstraintType</td>
<td>System declaration</td>
<td>A BehaviorConstraintType in EAST-ADL can contain instantiation of other BehaviorConstraintTypes in the form of BehaviorConstraintPrototype. Different prototypes are binded by their binding parameters and events.</td>
</tr>
<tr>
<td>ConstraintBiningParameter</td>
<td>Parameter</td>
<td>A constraint binding parameter binds two or more behavior prototypes. For timed automata these are considered as global variables defined in the main declaration part of an UPPAAL model.</td>
</tr>
<tr>
<td>ConstraintBiningEvent</td>
<td>Channel declaration</td>
<td>All the EAST-ADL constraint binding events are mapped to channel declaration for synchronizing between different processes.</td>
</tr>
<tr>
<td>State</td>
<td>Location</td>
<td>Same concept</td>
</tr>
<tr>
<td>initState</td>
<td>Initial location</td>
<td>Same concept</td>
</tr>
<tr>
<td>timeInvariant</td>
<td>Location invariant</td>
<td>Invariants related to timing constraints in EAST-ADL and clocks in timed-automata.</td>
</tr>
<tr>
<td>Transition</td>
<td>Transition</td>
<td>Same concept</td>
</tr>
<tr>
<td>readEventOccurrence</td>
<td>Synchronization channel</td>
<td>While the channels are defined using the ConstraintBindingEvent, the usage is defined by the EventOccurrences. If a process is receiving a signal from a channel the channel name is suffixed with symbol '?'</td>
</tr>
<tr>
<td>writeEventOccurrence</td>
<td>Synchronization channel</td>
<td>Channel sending is suffixed with symbol '!'</td>
</tr>
<tr>
<td>timeGuard</td>
<td>Guard</td>
<td>This guard is only related to timing</td>
</tr>
<tr>
<td>quantificationGuard</td>
<td>Guard</td>
<td>All the guard conditions which are only specified by the quantification parameters correspond to quantificationGuard</td>
</tr>
<tr>
<td>effect</td>
<td>Update action</td>
<td>This triggers different kind of computations such as update of a value or generating an event. From EAST-ADL point of view, an effect is an occurrence of a transformation (or calculation).</td>
</tr>
</tbody>
</table>

Figure 5.5: Mapping scheme for the Temporal Constraint part of the Behavior Description Annex

a) For each state in EAST-ADL create a location in timed automata. The type of location is identified by the properties of an EAST-ADL state.

b) For each transition in EAST-ADL create a timed automata transition followed by the addition of guards and actions. For each guard and effect of an EAST-ADL transition, if an attribute has an associated port then declare it as a global variable. Similarly define a synchronization channel for each event occurrence.

2. For every behavior constraint prototype in the main behavior constraint type
define a timed automata process. The input parameters of the process is obtained from the binding parameters defined for the corresponding EAST-ADL behavior constraint prototype.

3. Generate system declaration for the main behavior constraint type.

5.2.2 Prototype Implementation

A prototype of the transformation has been developed for a proof of the concepts. For this prototype, an EAST-ADL meta-model is implemented in MetaEdit+ [83]4, followed by the implementation of the transformation using its native language called MERL (Meta Edit Report Language) [84]. The resulting output is an UPPAAL XML which can be directly input in the UPPAAL environment for analysis. The transformation has been demonstrated by a case study of a brake-by-wire system with and without taking slippery road condition into account. Fig. C.6 in Paper C shows an UPPAAL model after transformation from MetaEdit+.

5.2.3 Discussion

As shown in Fig. 4.6, a state in the EAST-ADL temporal constraint can be used to represent a mode, error state or a hazard. Hence, the BDA can be used for relating artifacts modeled by different EAST-ADL extensions. The transformation in turn enables a combined behavioral analysis for many purposes such as the development of a fault-tolerant system. This is illustrated to a limited extent in the appended paper C. A few issues which require consideration when using the transformation are as follows:

- The transformation only considers single data values. In other words an array of data is not considered. For the prototype, an arrayed data type is specified by a string variable. EAST-ADL provides a concept of composite data type which can be explored for further refinement of the algorithm.

- Additional semantics are introduced by certain timed automata tools such as urgent channels in UPPAAL. The transformation algorithm has to be altered to cater for such additional semantics. Along with the alteration of the algorithm, guidelines are also required for using EAST-ADL effectively. In this thesis, the information related to the type of channel like urgent or broadcast is added as comments associated with artifacts like read or write event occurrences of an EAST-ADL transition.

- The unit of time can be an issue while transforming EAST-ADL models to the tools for analysis based on timed automata. Depending on the tool support, a time value may be multiplied/divided by some factor. For example, the notion of clock in UPPAAL is an integer variable. For the transformations in this

4The meta-model development is a collaborative effort with the second author of Paper B.
chapter as well as the appended paper C 1\( \mu \)s was chosen as the smallest time unit and in EAST-ADL all the timing information was handled accordingly such as ms was defined as a constant value of 1000.

- For constant numbers, the transformation assumed that attributes defining the range of a data type, i.e., minimum and maximum are the same. In the recent advancements of EAST-ADL in the MAENAD project [79], the concept of ValueType has been added. This concept might be beneficial for the simplification of the technical implementation. This issue needs to be investigated and is left as a future work for this thesis.

5.3 EAST-ADL and AUTOSAR Relationship Investigation

In contrast to an earlier work [30], this thesis provides mappings for additional behavioral concepts like mode and mode group. In addition, the thesis also identified a few issues which need to be taken care of when refining an EAST-ADL architecture to an AUTOSAR realization.

As mentioned earlier in Section 3.4, a two-step approach is adopted for the investigation of the relationship between EAST-ADL and AUTOSAR. The first step is a bottom-up method where a position control and a fuel control systems are modeled in EAST-ADL using PapyrusUML modeler [95] for deriving a conceptual mapping. The position control system includes fundamental AUTOSAR artifacts such as software components, behavior specifications and an analysis model in Simulink. The fuel control system covers additional behavioral aspects especially related to the switching of operational modes. The fuel control and position control systems are SystemDesk [36] example cases from dSpace providing coverage of AUTOSAR artifacts sufficient to meet the thesis objectives with the scope defined in Section 3.2. As the second step which follows a top-down approach, a brake-by-wire (BBW) system is modeled in SystemDesk to validate the derived mapping scheme.

A primitive investigation was also carried out for automated transformation from EAST-ADL’s design level of abstraction to AUTOSAR [105]. In the investigation a Python script is generated from the AUTOSAR model using model-to-text transformation. For this purpose Acceleo [89], providing an implementation of OMGs MOF language for transformation of models to text was used. The Python script is used to access the SystemDesk API within its environment.

While the table in Fig. 5.6 summarizes the behavioral mapping, the table in Fig. D.1 of the appended paper D summarizes the mapping between the functional entities of EAST-ADL and AUTOSAR, i.e., which AUTOSAR elements typically realize an EAST-ADL element. The tables only describe the mapping scheme. For further information about the semantics, the readers are referred to EAST-ADL [119] and AUTOSAR [6] specifications.
Table 2. Mapping of EAST-ADL artefacts corresponding to AUTOSAR behaviour

<table>
<thead>
<tr>
<th>EAST-ADL</th>
<th>AUTOSAR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior::Mode</td>
<td>Mode</td>
<td>The concept is similar in both formalisms. A mode control can lead to the switches between different configurations or execution schemes</td>
</tr>
<tr>
<td>Behavior::ModeGroup</td>
<td>Mode Group</td>
<td>Same concept in both formalisms to organize a set of modes in a mutually exclusive group.</td>
</tr>
<tr>
<td>Behavior::FunctionTrigger</td>
<td>RTE Event</td>
<td>The execution behaviour of an EAST-ADL function is declared by function triggers. The type of corresponding AUTOSAR RTE Event is determined by the TriggerPolicy and the associated Event Function and constraint for the function trigger. TriggerPolicy=Time implies a periodic event. (Timing event in AUTOSAR) TriggerPolicy=Event implies other events.</td>
</tr>
<tr>
<td>Timing::EventFunctionClientServerPort</td>
<td>Operation Invoked Event and Asynchronous Server Call Returns Event</td>
<td>An event function is associated with a function type in EAST-ADL. The type of event function determines the type of RTE Event in AUTOSAR EventKind = receiveRequest implies Operation Invoked Event EventKind = receivedResponse implies Asynchronous Call Returns Event</td>
</tr>
<tr>
<td>Timing::EventFunctionFlowPort</td>
<td>Data Received Event</td>
<td>The port considered in this kind of event should have the value 'IN' for its 'Direction' property.</td>
</tr>
</tbody>
</table>

Figure 5.6: Mapping of EAST-ADL artifacts corresponding to AUTOSAR behavior

5.3.1 Mapping Illustration

An illustration of the mapping is shown in Fig. 5.7.

Figure 5.7: An illustration of EAST-ADL and AUTOSAR mapping

In Fig. 5.7 the elements above and below the thick horizontal line belongs to
EAST-ADL and AUTOSAR respectively. It can be seen that an EAST-ADL design function type or its specializations can be mapped directly to an atomic software component or its specializations. The same applies for AUTOSAR’s RTE events which can realize EAST-ADL EventFunctions. However, for the case of runnables, there exist \( n \text{-to}-m \) relationship. This implies that a runnable can realize multiple elementary design function types. The same applies for the realization of composite design function types by atomic software components. In case of execution behavior, the artifacts from EAST-ADL’s timing model can be mapped to AUTOSAR behavior. For logical behavior, there exist at least two possibilities. One possibility shown in Fig. 5.7 is the use of EAST-ADL’s external behavior representation support (‘Behavior::FunctionBehaviorKind’ in Fig. 4.2). In this case behavior of an EAST-ADL function is specified using tools like Simulink followed by automatic code generation and associating the generated code with an AUTOSAR runnable realizing the design function type. The second possibility is to generate the code directly from EAST-ADL native behavior specifications, i.e., the behavior description annex to generate the code.

### 5.3.2 Additional Observations

The factors which may affect the decisions required for refinement of software architecture from EAST-ADL to AUTOSAR are as follows.

#### Mode Switch Port and Event

Modes are declarative in EAST-ADL. Mode change and related communications, e.g., mode switch event are not part of the EAST-ADL specifications. To handle this we can assume an EAST-ADL FunctionFlowPort to be equivalent to a mode switch port provided that it is referred to in the function trigger properties in addition to the event condition for activation. This is illustrated in Paper D.

#### Data Types and Prototypes

EAST-ADL only supports basic data types including Boolean, float, integer, string as well as composite data types. As a part of defining an AUTOSAR software architecture, these abstract data types are required to be mapped to concrete implementation data types with signedness, number of bits, coding, etc. Furthermore, in contrast to AUTOSAR, only one data type can be assigned to a single port in EAST-ADL. This is handled by the use of port groups. Depending on the type of realization, i.e., an atomic software component or a runnable, a port specifies an inter or intra software interaction through an RTE event or an internal variable respectively. This is described and illustrated in Fig. 5.7. It should also be noted that there is a DataElementPrototype for every EAST-ADL port. The data types should have matching specifications. Several EAST-ADL ports may be

---

\(^5\)an elementary design function type has the value of property \texttt{isElementary=TRUE}\n
aggregated in a single AUTOSAR interface. EAST-ADL port groups are candidates for aggregation depending on connection patterns.

Design Function Type Realization

For simplicity a one-to-one mapping between a design function type of EAST-ADL and an AUTOSAR runnable or software component is adopted in this thesis. This is determined by the property ‘isElementary’ of an EAST-ADL function type. There also exist direct correspondence between specialized function types in EAST-ADL and software components in AUTOSAR. For example, a local device manager in EAST-ADL can be realized by a sensor / actuator software component in AUTOSAR. In actuality there is a potential n-to-m relationship between EAST-ADL function types and AUTOSAR runnables / software components as shown in Fig. 5.7. This implies that several design function types can be realized by one runnable or vice versa. The same applies for the realization of a design function type by AUTOSAR atomic software components.

5.4 Discussion

This chapter summarized the work for integrating EAST-ADL with timed automata and AUTOSAR. These efforts cover two different aspects of integration, i.e., horizontal (at the same abstraction level) and vertical (across abstraction levels). Due to the n-to-m mapping possibilities between an AUTOSAR software component / runnable and EAST-ADL design function types, the selection of the right heuristics is a challenging task. To complement the work on the integration of EAST-ADL and timed automata, the thesis has also explored different possibilities to model and simulate advanced features. The next chapter deals with dynamic configuration mechanisms.
Chapter 6

Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

This chapter deals with the research question 1c (Section 3.3) and presents an evaluation of EAST-ADL and its integration with timed automata for dynamic configuration mechanisms. In particular relocation of a software component from a failed node to another on occurrence of a failure is considered. The relocation of a software component can also be carried out for optimizing system resources as well as managing a new device. The scenario chosen for this part of the work is the generic use case 3 of the DySCAS project discussed in Section 2.1.

The chapter is organized as follows. First the chosen middleware, i.e., DySCAS [27] is introduced. The DySCAS middleware system is developed with consideration of the state of the art technologies like CORBA, DynamicTAO, policy based computing, etc. A detailed description of the scenario under consideration, model simplifications are presented in Section 6.2. In Section 6.3 the results of model checking are discussed. Experiences and observations made during the work are presented in Section 6.4 followed by a concluding discussion in Section 6.5.

6.1 DySCAS Middleware Architecture

This section presents a conceptual overview of the DySCAS middleware architecture. For detailed specifications the readers are referred to [27].

As shown in Fig. 6.1, DySCAS follows a well-defined strategy in terms of data and control flow. A platform is used to refer to the software and hardware which support the execution of applications and platform such as microprocessors, operating systems, communication networks like CAN [51]. The platform communicates with the application programs, the external devices as well as the middleware services for controlling the execution/triggering of the tasks.
As opposed to a statically configured embedded system, which also contains the elements belonging to the three blocks on the right of Fig. 6.1, the left hand part constitutes elements which are able to perform run-time reconfiguration. Dynamic allocation and addition of application software is an example of such reconfiguration.

### 6.1.1 Core Services

The core services are the major elements providing the support for reasoning and decision making for dynamic configurations and quality control. The core services and their functionalities are as follows:

- **Resource Deployment Management Service** (RDMS) supports resource and execution control as well as software loading on a networked system platform. Most of the lower level operations are carried out by this service. Apart from the context monitor, external device control and application and resource control are the two types of internal computational modules.

- **Dependability & Quality Management Service** (DQMS) is responsible for on-line dependability control and QoS based optimizations. Quality control is a module for performing computations within a DQMS.

- **Autonomic Configuration Management Service** (ACMS) supports the deduction of dependencies between different components for overall system. It also works as a planner for re-configurations. The computational modules related to ACMS are the task scheduler and configuration resolver.

- **Autonomic Configuration Handler** (ACH) is the coordinator for the configuration operations scheduled by the ACMS.

- **Repository Service** supports storage, maintenance, and retrieval of files, configuration rules, component images, and logging of runtime information.
6.1. DySCAS Middleware Architecture

- **Software load management service** works closely with the repository service. It is responsible for executing the software load operations.

The above mentioned service can be both global and local forming a hierarchical structure. As the name indicates a global service is responsible for global level (i.e. whole system) activities and decisions. For example, maximum number of applications on a node. In contrast a local service is related to node level activities and decisions. For example, selecting the schedule for the execution of a set of applications running on the node.

6.1.2 Core Service Structure

As shown in Fig. 6.2 each core service has the following three different types of internal modules:

- **Context manager** module responsible to receive, derive and disseminate context information.
- **Computation module** for performing computations and making decisions based on the information available from the context manager module. The decision functions can be both static such as predefined algorithms or dynamic which can be changed at run-time such as policy based mechanisms [3]. A component can have multiple computation modules.
- **Execution controller** which controls the behavior of a component.
6.1.3 Signal and Data Types

The signals, data and the input and output ports of the basic core service structure are classified into the following five types:

- **Service requests** related to the operations / decisions performed by a specific component.
- **Service feedbacks** related to the results of the service requested.
- **Context publication/subscription** for measured or derived context information. A new service can publish (subscribe) the availability of the context provided (required) by it.
- **Context information notifications** related to the changes of context information. This refers to the update of an already existing context published or subscribed by one or more services.
- **DySCAS events** such as error signals, signals related to component initialization and requests for change of mode.

6.1.4 DySCAS Behavior

DySCAS [27] specifies several kinds of behavior. For the execution controller of a core service the behavior includes change of modes (configuration, run, error, wait and shutdown) based on middleware events, reading and writing on the input and output queues, and invoking the internal computation modules based on the type of received signal. A simplified representation of the execution controller in the form of a state machine is shown in Fig. 6.3\(^1\).

The DySCAS specifications also specify the behavior of different components of the middleware. One example is the system startup process shown in the Fig. 6.4. BCMS in Fig. 6.4 refers to the Basic Communication Management Service\(^2\).

6.1.5 Interface Services

The interface services provide support for the interactions between the middleware and its environment such as applications and hardware platform [27] (also refer to Fig. 6.1). There exist two types of interface services namely application and instantiation. The instantiation interface is further divided into the following services:

- **Master and Slave Configuration Management Services** are the services for controlling the execution of different services during the startup phase.

---

\(^1\)The simplification also includes exclusion of some of the state transitions

\(^2\)A basic communication service is an optional service for transparent communications [27]
6.1. DySCAS Middleware Architecture

Figure 6.3: Execution controller for a DySCAS core service [27].

- **Node Handler Services** for controlling the platform parameters such as operating frequency, devices, etc.

- **Basic Communication Service** is an optional service for transparent communication between the middleware and application programs.

- **Time Service** is an optional service for synchronization at a global level using synchronized clocks or a shared global clock. The functionalities include timing or clock information, clock set / reset, etc.

- **QoS Measurement Service** is an optional service for handling QoS related values. The functionalities can include advanced QoS data generation, measuring and processing of QoS data. Every core service is further divided into local and global service and is derived from the same basic structure as shown in Fig. 6.2.

### 6.1.6 Deployment Strategies

Due to the heterogeneity of the target platforms such as AUTOSAR and OSGi, a DySCAS middleware can be deployed in two different ways [27]. The first method is...
Chapter 6: Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

by the introduction of an abstraction layer between the platform and the DySCAS middleware. The role of the abstraction layer is to handle the portability issues by separating the details related to the platform. This in turn requires a software interface for mapping and transformation of services. The second approach is the usage of a dedicated platform specific implementation of the DySCAS middleware services. While the first choice has overhead in terms of performance and memory, the second approach can be more complicated.

6.2 The Scenario Under Consideration

The chosen scenario for this work is related to the relocation of a function from one processing node to another in case of hardware failure. A major assumption based on the experiences from the earlier work described in Chapter 2 and [98] is that all the relocatable functions or software are already present on the prospective ECUs. The reason is that many of the simulation platforms such as TrueTime, and operating systems like the one used in [98], do not have support for dynamic linking and loading of software.

The chosen scenario specifically covers the third DySCAS use case, i.e., GUC3 related to closed reconfiguration mentioned in Section 2.1. However, depending on the need and operational situation, a similar kind of dynamic configuration can
be considered for GUC4 related to optimization in the form of load balancing for better system performance.

With focus on network communication and configuration logic, the setup chosen for this part of the work is inspired by the work presented on graceful degradation of a driver assistance system in [55] and the EBS system presented in [85]. The overall system topology is shown in Fig. 6.5 and can be divided into the following four parts:

- EBS system consisting of an adaptive cruise control (located on the ACC ECU), EBS (located on the EBS ECU) and brake (located on the Brake ECU). The three components are the same as presented in [85].

- The radar part of the EBS system in [85] has been replaced by two additional functions namely Raw Data Sensor and Object Detection. The raw data sensor corresponds to a LIDAR (Light Detection And Ranging), a laser based object detection device. The two functions are inspired by the scenario of graceful degradation for driver assistance systems presented in [55].

- DySCAS middleware services for managing the reconfiguration and startup mechanisms. The services are distributed over several ECUs.
Chapter 6: Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

- The hardware such as transceiver and network (CAN Bus in Fig. 6.5).

The dark black line in Fig. 6.5 shows a network through which 6 ECUs are connected with each other.

### 6.2.1 Application Components

The applications and their respective timing constraints are as follows:

- **Raw Data Sensor** - The task of this function is to send the received raw data from a sensing device like LIDAR or RADAR to an object detection function. This function runs periodically with a period of 100µs where the minimum and maximum execution time is 30µs and 40µs respectively.

- **Object Detection** - This function runs on one ECU and can in case of a failure be shifted to another ECU. It is responsible for detecting objects based on the raw data received by it. Usually such functions are implemented using algorithms like a Kalman Filter on devices like FPGAs. This function runs periodically with a period of 10ms seconds where the minimum and maximum execution time is 1ms and 2ms respectively.

- **Adaptive Cruise Controller (ACC)** - The task of this function is to decide if an object detected is within or outside a risk zone. A risk zone is referred to as the zone where a collision with another vehicle is probable. This function runs periodically with a period of 10ms seconds where the minimum and maximum execution time is 1ms and 3ms respectively.

- **Emergency Braking System (EBS)** - An EBS system based on the information from ACC and other factors such as vehicle speed, etc. decides if an emergency braking is needed or not. In case an emergency braking is required a signal is sent to the brake function. This function runs periodically with a period of 10ms where the minimum and maximum execution time is 200µs and 5800µs respectively.

- **Brake** - As the name indicates, the task of the brake function is to interact with the brake actuators to execute the braking requested by the EBS function. This function runs periodically with a period of 5ms seconds where the minimum and maximum execution time is 50µs and 500µs respectively.

### 6.2.2 Middleware Services and the Deployment Strategy

As discussed in Section 6.1.6, there are two possible deployment strategies which can be adopted. The strategy followed for this scenario is a platform specific implementation. The DySCAS services are implemented only on the three ECUs which are involved in the dynamic configuration scenario, i.e., shifting of the Object Detection function to the Sensor ECU in case of a failure of the Object
6.2. The Scenario Under Consideration

Detection ECU. The Global Services ECU hosts the middleware services like master configuration management service for global level decisions and configurations. The model after modifications is shown in Fig. 6.6.

![Diagram of the scenario](image)

Figure 6.6: Dynamic configuration setup after modifications

The abbreviations in Fig. 6.6 are as follows:

- LACMS - Local Autonomic Configuration Management Service
- MCMS - Master Configuration Management Service
- GACMS - Global Autonomic Configuration Management Service

6.2.3 Modeling Simplifications

The targeted setup is subject to the following simplifications. The main simplification is to remove all the DySCAS core services which do not play any role in the scenario under consideration. The other simplifications are as follows:

1. For the sensor and object detection ECUs, the DySCAS middleware consists only of their respective Local Autonomic Configuration Management Services. This implies that the services which are not being used in the scenario are out of scope.
2. For the global services only the Master Configuration Management Service is considered. Furthermore, the transceiver for the Global Services ECU is also excluded.

3. All the DySCAS services are modeled as aperiodic instead of periodic. Similarly, all the services of the instantiation interface are treated as one. This assumption is made to avoid modeling overhead.

4. The effects of shared processing resources such as memory and processing power are not considered.

5. The ID of each message is 4 bits whereas the details of communication are as follows:
   a) The message lengths are 32, 16, 64, 32, 4 bits for Sensor ECU, Object Detection ECU, ACC ECU, EBS ECU and Brake ECU respectively.
   b) With 1 as the highest, the priorities given to the messages from each ECU are as follows. Brake = 1, Object Detection ECU = 1, Sensor ECU = 2, EBS ECU = 3, ACC ECU = 4.
   c) The time taken for transmitting a bit is 4\( \mu s \).

6. It takes at least 1\( \mu s \) to enable a function on a node.

6.2.4 Model Overview and Snapshots
As mentioned earlier, the scenario was modeled with both EAST-ADL and timed automata using MetaEdit+ and UPPAAL. The following presents an overview of different functions and their snapshots. Due to similarity in appearance, only MCMS model is chosen for its representation in both EAST-ADL and timed automata.

MCMS
The MCMS design can vary dependent on the platform on which it is running. The modeled behavior for the MCMS is shown in Fig. 6.7. Initially the startup of a system is triggered with the ‘enableObjNormal’ event. The event ‘FailureOccur’ is received by a separate process developed for simulation purpose only. The location ID of the node on which the object detection function resides is 1 at the beginning of simulation. On occurrence of a failure, the new location, i.e., ID = 0 along with the event ‘enableObjNormal’ is broadcasted to activate a new configuration. The events are received by LACMS. The urgency in EAST-ADL is specified by setting the property ‘isLogicalTimeSuspended’ of the state invariant as specified for the state ‘CurrentObjectDisable’.
6.2. The Scenario Under Consideration

Figure 6.7: MCMS behavior implemented in UPPAAL (above) and MetaEdit+ (below)

**LACMS**

Fig. 6.8 shows the behavior of LACMS. The logics are as follows. The LACMS receives a request for enabling (‘EnableRequestIn’ in Fig. 6.8) or disabling (‘DisableRequestIn’) a function along with the ID (‘currentObjectLocation’) of the node on which the function is to be enabled / disabled. If the ID of the node (‘SelfId’) on which a particular LACMS resides and the requested ID matches then a transition from the initial state to enabled or disabled states (‘LACMEnableFunction’ and ‘LACMDisableFunction’ in Fig. 6.8) occurs. For enabling a function a 1µs time is specified. The enabling is carried out by sending an event ‘EnableObjectOnNode’ to the application (Object detection function for the scenario under consideration).
Chapter 6: Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

Figure 6.8: LACMS behavior implemented in EAST-ADL

**Object Detection Function**

Fig. 6.9 shows the modeled behavior of the object detection function. The ‘enable’ and ‘disableOrFailure’ events correspond to the events received from the LACMS for enabling and disabling the function. The bottom three states follow the same pattern as that of a periodic function shown in Fig. 5.1.

**Raw Data Sensor**

The raw data sensor is a periodically executing function having no additional logics therefore, its model is the same as shown in Fig. 5.1a.

**Instantiation Service**

The role of the instantiation service is to provide an interface between the hardware and the applications or middleware services. The instantiation services in the modeled scenario is responsible for managing network signals sent by different applications on a node. Fig. 6.10 shows the model of the instantiation service of the sensor ECU in timed automata. The actions ‘TransRequest’ and ‘TransComplStatus’ are used to communicate with the transceiver. The other two actions namely ‘DataToBeSend’ and ‘DataToBeSend2’ corresponds to the data from the two applications, i.e., Raw Data Sensor and Object Detection function (ODF).
6.2. The Scenario Under Consideration

The raw data sender is sending data only to the ODF. A network transmission only occurs if the ODF on the sensor node is enabled.

Functional Structure

A subset of the functional structure is shown in Fig. 6.11. All the three functions are design function prototypes but typed by different function types. For example, the bus and the transceiver correspond to hardware and are thus typed as hardware function type. On the other hand EBS is an application modeled by generic design function type in EAST-ADL. Function flow ports are used for communications between different functions.
Chapter 6: Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

6.2.5 Timing Constraint

Only one constraint, i.e., the time to start a software at a new location should not be more than $10\mu s$ after the detection of a failure, is chosen for the scenario.

6.3 Verification

6.3.1 Verified Properties

The following three properties are to be verified:

1. The two object detection functions are never enabled at the same instant of time. This property is used to verify the correctness of the logics modeled for the application.

2. The system is deadlock free. EAST-ADL by its semantics is deadlock free. A system with a deadlock condition can indicate errors in modeling including the possibility of assigning wrong priorities or message lengths.

3. The time taken from the detection of a node failure to the startup of the software is not more than $10\mu s$. This property is the timing constraint specified in Section 6.2.5.
6.3. Verification

Figure 6.12: CAN transceiver [72] modeled in EAST-ADL

6.3.2 Verification Results

Out of the three properties the timing constraint of $10\mu s$ was not satisfied. The main reason is that no timing constraint is assigned on the event related to the activation of the function after relocation. In other words, there is no invariant defined for the location labeled ‘NewLocationEnable’ shown in Fig. 6.7. To handle this issue either the timing constraint has to be relaxed or the constraint on execution of the applications or middleware services has to be modified.

Analyses such as the one discussed above can be useful for many cases.
Chapter 6: Evaluating EAST-ADL and Timed Automata Integration for a Dynamic Configuration Scenario

generic usage can be in a situation where execution properties of applications running on the platform are specified by one design team and the constraints on the overall system are specified by another team such as an OEM (Original Equipment Manufacturer). Specifically for DySCAS, such kind of analysis can be useful in analyzing timing constraints related to the middleware events before implementing it on a real platform.

6.4 Experiences and Observations

Based on the experiences from the work discussed in this chapter, study of the EAST-ADL specifications [27] and consideration of the requirements listed in Section 2.2.2, the following observations are made.

• As mentioned in Section 2.2.4, modes, state-machine behavior and computations are common for static and dynamic configurations. It is possible to specify both operational modes and state-machine type behavior in EAST-ADL. This can be done by using its Functional Behavior package for system configuration modes, and temporal constraints part of the behavior description annex for specifying logical behavior and computation constraint part of the BDA for other necessary causal constraints on the computation control flow. The work presented in this thesis mainly uses the temporal constraints for modeling state machine type behavior.

• In Section 1.1.1, the term self-management was introduced. Dynamic configuration is only a subset of self-managing features. EAST-ADL supports specification of configuration variations at design time. Run-time variations are supported to a limited extent through internal functional behavior and activating or deactivating a function based on mode changes, etc., as illustrated in this chapter. The usage of EAST-ADL and its possible extensions for supporting self-management features requires further investigation.

6.5 Discussion

This chapter presented the work on validating the integration of EAST-ADL and timed automata in the context of a dynamically configurable system. The behavioral aspects which are common to both existing and advanced systems envisioned for the future can be modeled using EAST-ADL. This implies that logical and execution behavior can be modeled using EAST-ADL and analyzed using timed automata. However, some features like the run-time configuration specification of a product as a whole and its analysis are still open issues.
Chapter 7

Discrete Event Modeling and Simulation Tools: A Comparison

This chapter is related to the research question 3 (Section 3.3) and corresponds to the arrow 3 in Fig. 3.2. The main objective is to explore the extent to which Stateflow and SimEvents models can be used interchangeably for simulating and analyzing a system behavior. The arrow 4 in Fig. 3.2 is also indirectly addressed by inferring the relationship between EAST-ADL, Stateflow and SimEvents from the results. It is assumed that the readers have a good overview of both Stateflow and SimEvents\(^1\).

7.1 Stateflow and SimEvents Comparison

In previous work [103] three tools namely, Stateflow, SimEvents and TrueTime were used for modeling and simulation of various dynamic configuration scenarios. All the three tools were found to be satisfying most of the requirements derived for modeling and simulation tools in [103, 104]. This thesis is focused on EAST-ADL and in particular the timing model (TM) and the behavior description annex (BDA). Both TM and BDA, with the exception of the attribute quantification constraints of BDA, are targeting some form of discrete event behavior. Therefore, only Stateflow and SimEvents are chosen for this work. Both Stateflow and SimEvents can be considered as a tool as well as a language.

The roles of Stateflow and SimEvents as described by Mathworks are as follows [59]:

- SimEvents can be used to model activities within a system by entities and their movement. It is possible to generate events independent of the time steps corresponding to the ODE solver.

\(^1\)Interested readers can refer to Section 4.2.1, http://www.mathworks.com/products/simevents/ and http://www.mathworks.com/products/stateflow/
• Stateflow can be used to graphically model states of a system and the conditions on which the system transits from one state to another. While SimEvents depicts movement of data, Stateflow shows transitions.

Despite the above mentioned description, there exists a need for evaluating the two tools for usage as alternatives for modeling a system behavior. Especially, in the context of the EAST-ADL behavior description annex, such an evaluation can be the first step to find possibilities for transforming the temporal and computation constraint models to SimEvents and Stateflow models.

7.1.1 Comparison Method

An automatic drivetrain [60] is chosen as the main case study. The chosen system is one of the example systems for Stateflow provided by Mathworks. It features characteristics such as mode changes which are common for both statically and dynamically configurable system. In terms of the DySCAS middleware, which has been the targeted dynamically configuring system, a state transition representation can be used to model configuration or application modes, decision functions and context information. In an earlier work discussed in Chapter 2 and [103], a state transition representation was used to model quality of service (QoS) levels of different applications, context information in terms of varying resource consumption and decision functions for allowing or rejecting a new software based on the available resources and security authentication results. The computation can be related to calculations for context derivation as well as prediction of behavior. In the earlier work, computations were implemented as Simulink functions for calculating QoS levels using algorithms presented in [44].

![Figure 7.1: Top level view of Stateflow and SimEvents models](image_url)

For the comparison between Stateflow and SimEvents, the gear controller of the automatic drivetrain system was remodeled using SimEvents. As shown in 7.1,
the two models were provided with the same inputs for comparing the output response. Based on the observations from modeling the drivetrain as well as modeling of different dynamic configuration scenarios [103], a comparison is made. The comparison also considered a subset of the framework in [41] for comparing different model-based approaches for embedded systems development.

Figure 7.2: Shift logic in Stateflow [60]
7.1.2 Results

Fundamental Comparison

The following are the fundamental differences and commonalities between Stateflow and SimEvents.

- An *event* in Stateflow and SimEvents can refer to a function call, or to a change in a data value from low to high and vice versa. In SimEvents, advancement of an entity and completion of a service by a server are also considered as events.

- A *state* in Stateflow can represent a particular condition or mode of system. For SimEvents, a state can correspond to the location of an entity or an ongoing operation such as processing of an entity by a server, etc.

- A *transition* in Stateflow is the point at which a system changes from one state to another. At a transition, an action such as generation of events or some calculations can be triggered. The occurrence of a transition can be restricted by applying guard conditions in the form of events or values of data. For SimEvents, a transition can correspond to movement of an entity. The movement of an entity can be utilized for generation of events or manipulation of data. The movement can be restricted / redirected by specifying conditions in the path using appropriate function blocks.

Transformation of Stateflow Models to SimEvents

Stateflow models can be transformed to SimEvents models as follows:

1. For each Stateflow chart a SimEvents subsystem can be created.

2. If the chart is triggered periodically then for each parallel state a time-triggered entity generator is added in the corresponding SimEvents subsystem. This is illustrated by the generators named ‘selection_state’ and ‘gear_state’ in Fig. 7.6 (compare with Fig. 7.2). For an event triggered chart an event triggered entity generator is added.

3. For each data input of the Stateflow chart, an attribute is assigned to the generated entities in SimEvents. This is shown by the block ‘Set Attribute for Input Ports’ in Fig. 7.6.

4. For a function called within a parent state, a function call subsystem is added to the SimEvents model. This is shown by the subsystem ‘Compute Threshold’ in Fig. 7.6. To trigger the event a function call generator is added before the outputs of the subsystem are used. This is shown by ‘Entity Departure Function-Call Generator’ in Fig. 7.6. The outputs of the function-call subsystem are added to the entity using a set attribute block. This is shown by ‘Set Attribute 1’ in Fig. 7.6.
5. The state logic of Stateflow chart is implemented using a SimEvents Attribute Function\(^2\) block. This is shown by the ‘selection_state_logic’ block in Fig. 7.6 which corresponds to the parallel state labeled as ‘selection_state_logic’ in Fig. 7.2. The state logic can then be implemented using conditional statements. Apart from the input and output variables, the old state value is also one of the inputs to the attribute function. Furthermore, the states have to be represented by integer numbers due to the fact that a string variable cannot be assigned to attributes in the current version of SimEvents. The logic implemented for the ‘selection_state’ of the Stateflow Chart in Fig. 7.2 is listed in Fig. 7.3

```matlab
function [out_currentState, out_wait, out_updown] = 
fcn(VehicleSpeed, down_th, up_th, currentState)
    if currentState==1 && VehicleSpeed<down_th
        out_currentState=2;
        out_wait=0.04;
        out_updown=0;
    elseif currentState==1 && VehicleSpeed>up_th
        out_currentState=3;
        out_wait=0.04;
        out_updown=0;
    elseif currentState==2 && VehicleSpeed<down_th
        out_currentState=1;
        out_wait=0.04;
        out_updown=0;
    elseif currentState==2 && VehicleSpeed<=down_th
        out_currentState=1;
        out_wait=0.08;
        out_updown=1;
        elseif currentState==3 && VehicleSpeed<up_th
        out_currentState=1;
        out_wait=0.04;
        out_updown=0;
    elseif currentState==3 && VehicleSpeed>=up_th
        out_currentState=1;
        out_wait=0.08;
        out_updown=2;
    else
        out_currentState=1;
        out_wait=0.04;
        out_updown=0;
    end
```

Figure 7.3: Selection state logic implemented as an Attribute Function in SimEvents

6. For transition delays and to cater for periodic chart update of Stateflow, additional timing information has to be added in the state logic. This

\(^2\)An Attribute Function block is similar to an Embedded Function block in Simulink
is shown by the ‘out_wait’ variable in Fig. 7.3. The Stateflow chart in Fig. 7.2 is executed every 0.04 sec and the delay between ‘down_shifting’ and ‘steady_state’ states is 0.04 sec. This is the reason for having the values of ‘out_wait’ being 0.08 for the case where transition is delayed. To simulate the delay, a single server\textsuperscript{3} is used as shown in Fig. 7.6. The variable for specifying the delay is used as the ‘execution time’ parameter of the single server.

7. The events in a Stateflow chart are required to be converted into their equivalent numbered data. This is shown by the ‘updown’ variable in Fig. 7.3. The ‘updown’ variable is later used as an input to a ‘Signal-Based Function-Call Generator’ shown in Fig. 7.6. This function call generator can then be used to generate a SimEvents entity using an ‘Event-Based Entity Generator’ (EBEG). The numbered data is assigned to the entity generated by the EBEG. The entity generated by the EBEG is combined with the entity of the state to which the Stateflow event is sent.

8. All the internal parameters are defined as constant inputs and treated as data inputs as in 3.

With this setup, the above mentioned mapping scheme is summarized in the table shown in Fig. 7.4.

Applying a Comparison Framework for Model-Based Development Approaches on SimEvents and Stateflow

In [41] a framework for comparing different approaches for modeling embedded systems is presented. The framework is divided into five categories namely; content, design context, analysis context, language and tool. Language and tool are the two categories chosen for comparing Stateflow and SimEvents. The comparison is as follows:

- **Comparison as Language**

  - Representation technique: In Stateflow a system is represented in the form of a hierarchical finite state machine (FSM). It is possible to model FSMs as both Mealy and Moore type FSMs. On the other hand a system in SimEvents is based on a discrete event model of computation represented by blocks of several kinds along with textual support for functional manipulations.

  - Adaptability: There exists no explicit support for modifying, extending, and scaling down for both Stateflow and SimEvents.

\textsuperscript{3}As the name indicates, a server in SimEvents is used to simulate a processing unit with features like preemption, estimation of CPU utilization, etc.
7.1. Stateflow and SimEvents Comparison

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Stateflow</th>
<th>SimEvents</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chart</td>
<td>Subsystem</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Chart trigger</td>
<td>Entity generator</td>
<td>The type of chart trigger determines the type of entity generator. For</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>periodic chart trigger a time-based entity generator is used. If a Stateflow chart is triggered by an event, the corresponding SimEvents system starts with an event based entity generator.</td>
</tr>
<tr>
<td>3</td>
<td>Event</td>
<td>Data, signal based</td>
<td>The data in SimEvents is used as the signal for generating an event.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>event generator and</td>
<td>The reception of event in turn corresponds to the generation of an entity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>event based</td>
<td>and assigning relevant attributes to it.</td>
</tr>
<tr>
<td>4</td>
<td>Data</td>
<td>Entity attribute</td>
<td>An attribute is assigned or read for input and output data. For internal constant parameters of Stateflow, a Simulink constant is used to set attribute to entities.</td>
</tr>
<tr>
<td>5</td>
<td>State logic</td>
<td>Attribute Function</td>
<td>All the state logical is specified as Matlab code in the attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>function.</td>
</tr>
<tr>
<td>6</td>
<td>Time delay</td>
<td>Server</td>
<td>The time delay becomes the execution time of the server.</td>
</tr>
</tbody>
</table>

Figure 7.4: Stateflow and SimEvents Comparison Summary

- Multi-views: Both SimEvents and Stateflow only represent a single view for a system under consideration.

- **Comparison as Tool**
  - Availability: Both SimEvents and Stateflow require licensing.
  - User interaction: Being based on Simulink both tools have a graphical user interface.
  - Code generation: It is possible to generate executable code from Stateflow. This is not possible with the studied version of SimEvents.

Efforts were made to remodel existing SimEvents models using Stateflow. It is found to be a non-trivial task unless and until some definite usage practices or heuristics are defined.

7.1.3 Limitation and Observations

A major limitation of the work is that it only considers a single layer of Stateflow charts. This implies that sub-charts are not considered. A judicious reader
can observe that interaction between multiple state charts is not included in the transformation described in Section 7.1.2. This can easily be dealt with in a similar way as for two parallel states of a Stateflow chart. Two Stateflow charts can communicate with each other either in the form of data transfer or events through ports. For communication in the form of data the approach mentioned in the 3rd bullet of the transformation can be utilized. For event based communication, an approach similar to the bullet number 7 can be adopted. The following are the additional observations.

- Mealy type machines in Stateflow are preferable for transformation to SimEvents for avoiding modeling overheads.

- The output of a SimEvents block can be delayed to at least one sample time depending on the parameter selection of the entity generator discussed in the 2nd bullet of the transformation.

### 7.2 EAST-ADL Relationship with Stateflow and SimEvents

In the previous sections SimEvents and Stateflow are compared. Based on the results and the EAST-ADL specifications [119, 126], the following can be inferred:

- The syntax of the temporal constraint part of the behavior description annex is directly mappable to a Stateflow chart. The table in Fig. 7.5 shows a possible mapping.

As seen in Fig. 7.5 artifacts like states, transitions, actions (‘effect’ in terms of EAST-ADL), events, etc. are directly mappable. The row 9 of Fig. 7.5 is illustrated by the events ‘UP’ and ‘DOWN’ in Fig. 7.2. These events are sent from ‘selection_state’ to ‘gear_state’.

There might exist issues related to the semantics depending on the style of modeling followed by a designer. For example, in Fig. 7.5 a time guard in EAST-ADL is mapped to temporal logic in Stateflow. This mapping of time guard is not always valid. This is due to fact that time guard can represent an actual time whereas the temporal values related to commands like ‘after’ in Stateflow are reset after every transition.

If a timed automata based semantics is assumed for the temporal constraints then the results from the work [94] on transforming timed automata models in UPPAAL to Stateflow can be utilized. In [94] the transformation considers both the logics and clock.

In addition, a Stateflow chart requires triggering information. To acquire the triggering information, the properties of the behavior constraint containing the temporal constraint under consideration has to be checked. Specifically, the ‘constrainedFunctionTrigger’ and ‘targetedFunctionType’ shown in Fig. 4.5 are required to be checked. From these two properties, it is possible
### 7.2. EAST-ADL Relationship with Stateflow and SimEvents

<table>
<thead>
<tr>
<th>S.No.</th>
<th>EAST-ADL Temporal Constraint</th>
<th>Stateflow</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Temporal Constraint</td>
<td>Chart of type 'Mealy'</td>
<td>Computations can not be specified in a state of an EAST-ADL temporal constraint model. Computations are only triggered with transitions. Therefore, the chart type is Mealy.</td>
</tr>
<tr>
<td>2</td>
<td>State</td>
<td>State</td>
<td>Same concept</td>
</tr>
<tr>
<td>3</td>
<td>Initial State</td>
<td>State with a default transition</td>
<td>In EAST-ADL an initial state is specified by setting the property isInitial to true. A Stateflow state with a default transition corresponds to an initial state from where the state machine execution begins.</td>
</tr>
<tr>
<td>4</td>
<td>Transition</td>
<td>Transition</td>
<td>Same concept</td>
</tr>
<tr>
<td>5</td>
<td>Time Guard</td>
<td>Temporal logic</td>
<td>A time guard in an EAST-ADL transition specifies guards related to clocks. In case of Stateflow Temporal logics like 'after' and 'before' can be specified to specify timing conditions.</td>
</tr>
<tr>
<td>6</td>
<td>Quantification Guard</td>
<td>Guard</td>
<td>A quantification guard corresponds to all the guard conditions which are not related to timing.</td>
</tr>
<tr>
<td>7</td>
<td>Effect</td>
<td>Transition action</td>
<td>Same concept</td>
</tr>
<tr>
<td>8</td>
<td>occurredLogicalEvent</td>
<td>Transition guard</td>
<td>A logical event corresponds to conditions related to one or more data variables becoming true.</td>
</tr>
<tr>
<td>9</td>
<td>occurredExecutionEvent</td>
<td>Event guard and action</td>
<td>An execution event in EAST-ADL is related to the occurrence of events specified by the EAST-ADL Timing Model such as EventFunctionFlowPort. In case of Stateflow, an event sender has to specify the receivers name.</td>
</tr>
</tbody>
</table>

Figure 7.5: Relationship between EAST-ADL and Stateflow

to find the timing properties specified by the Timing Model extension of EAST-ADL.

- The computation constraint of the behavior description annex can potentially be mapped to SimEvents. For example, a ‘logicalPath’ which “is a set of restrictions on the cause-effect flows of some observable logical and executional events” [126], can correspond to a combination of an entity generator, an attribute function and other entity handling and routing blocks. In this case an entity generator can be associated with a cause / event and the attribute function associated with the effect. The effect in EAST-ADL is
termed as ‘LogicalTransformation’. The thesis has mainly focused on the temporal constraint part of the BDA, therefore, the computation constraints are not covered in detail.

7.3 Discussion

This chapter presented a comparison of Stateflow and SimEvents as a basis for evaluating their usage as alternatives for analyzing EAST-ADL behavior description annex. Being based on finite state machines, the temporal constraints of the BDA can be directly mapped to Stateflow. Issues related to timing might arise which require consideration while analyzing EAST-ADL specifications using Stateflow. Further investigations are required for transforming BDA models to SimEvents.

A few important aspects related to modeling and simulation formalisms and tools for embedded systems development are usability, comprehensibility and granularity of models. A combination of Stateflow and SimEvents can be considered for platform based design. In this case, SimEvents can be used for modeling the platform such as processors and queues whereas Stateflow models can be used for modeling software logic. Varying levels of granularity for different parts of a system can also be considered. For example, an analysis for a distributed system with multiple microprocessors can be carried out for the effects of network. In this case, it should be more feasible to abstract the application logic with a simple state machine and a detailed network behavior with SimEvents.
Figure 7.6: Gear control in SimEvents
Chapter 8

Discussion

This chapter revisits the research questions posed in Section 3.3 and discusses the limitations and validity of the results. A few additional observations made during the course of the research work are also presented.

8.1 Results Review, Limitations and Validity

8.1.1 Research Question 1

How can we integrate EAST-ADL with timed automata based tools for a useful analysis of architectural specifications?

• Question 1a: What are suitable syntactical and semantical mappings between timed automata and EAST-ADL’s timing model and behavior description annex?

Answer: The timing model of EAST-ADL and timed automata can be related to each other by heuristics. A heuristic based approach can result in several different mapping schemes depending on the underlying assumptions, level of abstraction for analysis (e.g., inclusion or exclusion of operating system characteristics) and the method for deriving the relationship. The derived mapping assumes concurrency in the execution of different EAST-ADL functions as specified in [119]. The methodology is described in both Section 3.4 and Papers A and B.

A template-based mapping scheme covering both syntactical and semantical mappings is proposed in this thesis. A template is defined for each artifact of the timing model as well as function executions. The are two kinds of templates for function executions depending on the type of trigger, i.e., time or event. The template-based approach is discussed in Section 5.1.3, Papers A and B.
Chapter 8: Discussion

The temporal constraint of the EAST-ADL behavior description annex has the same semantics as that of timed-automata. Therefore, the thesis provides a syntactical mapping and focuses on the development of an algorithm which can be used by imperative transformation languages. The mapping of temporal constraints part of the behavior description annex is discussed in Section 5.2 and Paper C.

• **Question 1b:** Given a suitable mapping, is it possible to analyze an EAST-ADL timing model for consistency between its artifacts?

**Answer:** As mentioned in Section 1.2.6, consistency can be of several types. The thesis addressed the consistency within a model to check if the constraints specified on different parts of a system (e.g., braking and engine control) are consistent with each other. The checking of specification consistency between different timing model artifacts is demonstrated in Papers A and B.

Another way to check consistency of timing specification is by using the behavior description annex. For this purpose, the timing model artifacts can first be modeled using BDA followed by the transformation of the BDA model to a tool based on timed automata and perform model checking.

• **Question 1c:** Given a solution to Questions 1a and 1b, how can the solution be used for dynamic configuration mechanisms?

**Answer:** There are several possibilities to use the solution for advanced functionalities. In this thesis, a few aspects were evaluated with a scenario of relocation of software in case of a processor failure. As mentioned in Section 6.4, EAST-ADL can be used to specify dynamic configuration where flexibility is added to the application or the middleware behavior. The modeling of behavior can be carried out using the behavior description annex. Constraints on timing can be specified using the timing model. This can be followed by the transformation of the behavior and timing models for various kinds of analysis. In Chapter 6, verification of a few properties is illustrated. One property is the constraint on the delay between two events related to the occurrence of a fault and startup of software on a new node. Another property is the constraint on the activation of an application, i.e., two instances of the same applications are not enabled at the same instant of time. Similarly, in Paper C the usage of EAST-ADL and timed automata for developing a fault-tolerant system is demonstrated to a limited extent.

• **Limitations and Validation**

The EAST-ADL specifications [119] do not restrict the number of events associated with a function. However, the transformation of the timing model assumes only one data-input and one data-output event represented by ‘EventFunctionFlowPort’ in EAST-ADL for a function. Such data input and output events can be related to the data arrival at (or departure from) a port. The limitation of one input and one output event implies that the templates
are valid only for cases where there is a maximum of one input and/or output event. A single synchronization action in timed automata can also be used for multiple EAST-ADL events related to input or output of data through a port provided they occur at the same time. For the cases where the assumption is not valid, it is a trivial task to extend the templates. This can include additional behavior logics as demonstrated for the precedence constraint in Fig. B.7b in the appended paper B.

This thesis is focused on the transformation and conceptual mapping instead of the actual verifications in terms of model checking, therefore the scalability and verification of the models are not the focus. However, a few investigations were made by changing system configurations like addition of new timed automata processes, change of behavior, etc. Based on the observations, and the fact that timed automata models tend to have an infinite state space (which can be handled by partitioning into symbolic state-space) [11], it is inferred that the scalability has a direct correspondence to the extent of complexity supported by the formalisms and existing tools.

To validate the mapping scheme, a two-step approach is used. For the EAST-ADL timing model and timed automata relationship, an existing UPPAAL model of an emergency braking system covering both functional and network behavior was modeled in EAST-ADL. This was followed by transforming a brake-by-wire (BBW) system (which is a representative case for EAST-ADL) to timed automata for validation of the derived mapping scheme.

The automated prototype implementation of the transformation has been validated by the transformation of known models from EAST-ADL to UPPAAL. However, the queries for verification were generated manually. The automated generation of queries is a trivial task where a transformation engine only needs to generate text as mentioned under ‘verification’ in Section 5.1.3.

8.1.2 Research Question 2

How are AUTOSAR and EAST-ADL related to each other from a behavioral point of view?

- **Question 2a:** What are the possible relationships between EAST-ADL and AUTOSAR architectural configurations?

  **Answer:** The relationship between AUTOSAR and EAST-ADL is summarized in Section 5.3 and presented in Paper D. In Section 1.2.1 logical and execution behavior is described. One of the major principles behind the development of AUTOSAR is “Cooperate on standards, compete on implementation” [6]. Due to this, the internal logical behavior of an application is out of the scope of the AUTOSAR standard, hence the focus on the execution behavior. The logical behavior in AUTOSAR is usually in the form of the executable code associated with an AUTOSAR runnable.
The execution behavior of an AUTOSAR software component has to a large extent a one-to-one correspondence with the execution behavior as defined in EAST-ADL. For example, an AUTOSAR RTE event can realize an EAST-ADL Event Function.

There are two possible ways to bridge the gap between logical behavior of AUTOSAR (i.e. the executable code associated with runnables) and the logical behavior in EAST-ADL. The first is by generating an executable code directly from EAST-ADL behavior specifications and associating it with the AUTOSAR software component or runnable corresponding to the EAST-ADL function for which the code is generated. The second possibility, as mentioned in Section 5.3.1 is through the external behavior representation support for EAST-ADL. This can be achieved, for example, by associating a Simulink model with an EAST-ADL function followed by generating an executable code from the Simulink model and associating the generated code with the AUTOSAR runnable.

- **Question 2b:** What are the possible concerns which need to be addressed while refining an EAST-ADL configuration to AUTOSAR?

**Answer:** Three issues are presented in Section 5.3.2 and Paper D. One issue is related to mode switching. AUTOSAR explicitly supports execution control based on the switching (e.g., entry or exit) of different operational modes. For example, with the mode switch event in AUTOSAR, it is possible to specify if a runnable is triggered when the mode is active, enabled or disabled. In contrast, EAST-ADL only specifies the modes in which a particular function is enabled. This can lead to a difference in the behavior specification and the actual behavior.

The second issue is related to the data types. EAST-ADL only specifies basic data types such as boolean, float and string. On the other hand, additional details like signedness, number of bits, and ordering (little endian, big endian) and so forth need to be specified while defining an AUTOSAR architecture. Therefore, detailed information related to the data type also needs to be specified while refining an EAST-ADL configuration to AUTOSAR parameters. If not handled properly this issue can lead to implementation inconsistencies.

The third issue is the n-to-m mapping possibilities between AUTOSAR runnables / atomic software components and EAST-ADL design function types and prototypes. All the three issues point to the need for guidelines which can be used for developing refinement of EAST-ADL models to AUTOSAR.

- **Limitations and Validation**

Similar to the work on the integration of EAST-ADL and timed automata, the investigation of the relationship between AUTOSAR and EAST-ADL...
was a two-step process. Different AUTOSAR models related to control system development were modeled with EAST-ADL to derive the mapping scheme. This was followed by modeling of a brake-by-wire (BBW) system with AUTOSAR using the derived mapping scheme. The BBW is the same one used for EAST-ADL and timed automata integration as well as in several EAST-ADL related projects like ATESTST2 [123] and MAENAD [79].

Both EAST-ADL and AUTOSAR have a very broad coverage and a large number of parameters. Therefore the work is limited to the selected parameters in AUTOSAR and their corresponding EAST-ADL artifacts. The AUTOSAR behavior parameters like exclusive areas and port argument values are not considered.

8.1.3 Research Question 3

How can the modeling formalisms related to Stateflow and SimEvents be used as an alternative for analyzing EAST-ADL based architectural specifications?

Answer: Stateflow and SimEvents represent two different formalisms and tools. In this thesis, efforts were put forth to find a bidirectional relationship between the two tools. In Chapter 7, a mapping from Stateflow to SimEvents is presented.

Possible relationships between SimEvents, Stateflow and EAST-ADL are also discussed in Chapter 7. The relationships are based on the experiences from the work in this thesis as well as an earlier thesis.

SimEvents has a large number of blocks in its library. Each component has different semantics depending on its chosen parameters. This implies that it has a very large semantic space. Hence it is non-trivial to transform SimEvents models to Stateflow.

- Limitations and Validation
  As mentioned in Section 7.1.3, a major limitation of the work is that it does not consider hierarchy of Stateflow charts. The mapping is derived by utilizing the experiences from modeling and simulation of several scenarios. A validation is made by observing the response of the original Stateflow model and the transformed SimEvents model using the same random inputs to both the models.

8.2 Additional Experiences and Observations

A lot of observations have been made during the thesis, many of which are related to the results and presented in their corresponding chapters. The following are a few additional thoughts.

- For implementing a transformation algorithm, MERL was found to be more efficient for prototype implementation as compared to the EMF based transformation. Due to its imperative nature, it is possible to write complex
transformations with less time compared to ATL. However MERL requires more time to figure a way to navigate through the source models. This is in correspondence with what Mens et. al. mentioned in [82], i.e., although a declarative approach is more promising, an imperative transformation (e.g., MERL) is suitable for cases where models are updated incrementally. This is due to the fact that an imperative approach gives a better control over the transformation operations [82].

- As mentioned in Section 7.2 issues related to semantics might exist when transforming the same model to different tools. Therefore there is a need to specify guidelines for using EAST-ADL to achieve consistency between the architectural specifications and analysis model. The issue of consistency can be dealt with in several ways. This includes, but is not limited to, the usage of a subset of EAST-ADL relevant for the tools under consideration and separate specification models for different kinds of analysis for example, one model for Simulink based analysis and another one for timed automata based analysis.

- Abstraction plays an important role in analyzing a system. While tools like SimEvents provide support for detailed modeling, tools like TrueTime abstract details such as queues. During an analysis it is desirable to choose a higher abstraction level for some parts and a detailed model for the other parts of the system under consideration. A library of different types of components with varying abstraction levels can be a possible solution to deal with variations of the desired granularities.

- The timing constraints specified by the EAST-ADL timing model can be analyzed in at least two different ways. The first is through model checking by transformation to a timed automata tool like UPPAAL. This can also be done using other modeling checking tools like SPIN [9] based on other formalisms but will require additional consideration for modeling timing. The second way is by modeling the constraints as monitors in tools like Simulink. In this case a monitor can be developed for example, using Stateflow where its temporal logics (e.g., specified by ‘after’ and ‘before’ functions) can be utilized for modeling the timing behavior. SimEvents can also be used where the timing constraints can be specified in the form of timeouts assigned to relevant entities. The two approaches, i.e., timed automata and simulations with Stateflow/SimEvents have their own pros and cons. State explosion problems are very likely with timed automata. On the other hand, the simulation based approach is useful if all relevant system inputs are captured in the simulations.

- A lot of efforts are required to model a complete system with EAST-ADL therefore, it might not be useful for small systems where the information is limited and the development is carried out by a small team. For large systems, this can be an issue in the initial phases related to the adoption
of the language but will be beneficial for large systems like automobiles in a longer term perspective.

- The complexity in terms of modeling and simulation increases with increasing amount of dynamism. In the context of EAST-ADL, a higher degree of dynamism will not only require artifacts from the behavior description annex, timing model and the core structure but also from the variability model to analyze a system. In case of formalisms and tools like Stateflow and SimEvents, the modeling may require additional features such as dynamic linking of different Simulink models during a simulation.

### 8.3 Overall Contribution towards the Advancement of Science and Technology

The main contributions of the thesis towards the advancement of science and technology is the investigation of relations and development of mappings between multiple formalisms and tools, paving way for a seamless engineering environment. In addition, the work on integrating EAST-ADL and timed automata was used for problem analysis and to provide feedback during the development of the EAST-ADL’s behavior description annex.

A significant part of the thesis is based on an earlier work (discussed in Chapter 2) which in turn was used for a) validation of the development of the DySCAS middleware and b) providing guidelines for algorithm design for developing dynamically configurable system using the DySCAS middleware [27].
Chapter 9

Future Work and Conclusion

The work in this thesis can be extended in several ways, for example:

- The thesis considered the timing model in EAST-ADL which is derived from the TADL [129]. TADL has evolved with several new artifacts [128]. The work can be extended by exploring the possibilities to formally represent the new artifacts with timed automata. Further refinements such as generalization of the rate transition templates can also be considered.

- Ptolemy [40] is one of the tools which supports multiple models of computation. In addition, there are also commercial tools like SCADE [42] being used in the industry. The work presented in this thesis can be extended by including these tools for modeling, analysis and model transformation.

- An important activity of embedded system development is testing. An investigation of the generation of test cases from the models specified using EAST-ADL behavior and timing extensions can be considered as one of the key future extensions of the thesis. In [54] test cases are generated from timed automata models. As the EAST-ADL temporal constraint is based on timed automata, the results of [54] can be utilized as a starting point.

- Modeling formalisms, tools and their integration are not sufficient to ensure efficient development process of a complex system. There is a need to develop methodologies which provide guidelines and rules for using different languages, tools and the integration solutions. An example of such methodology is provided for TADL [129] which covers all the development aspects from requirements to implementation in the context of EAST-ADL and AUTOSAR for timing constraints. Similar guidelines should be developed for behavioral aspects.

- Automated code generation from EAST-ADL behavior models can also be considered as one of the future possible directions for a closer integration with AUTOSAR.
• It is also interesting to evaluate how light weight or custom built architectures can be scaled to architectures like AUTOSAR and vice versa in the context of architecture description languages. From an EAST-ADL perspective, a few possibilities which can be explored are:
  
  – Abstracting an existing light weighted or custom built architecture at its design level of abstraction.
  
  – Issues in synthesizing architectural specifications into implementations of different scales.

• In terms of modeling and simulation, only a subset of possible reconfiguration mechanisms was addressed in this thesis. The work can be extended by exploring additional mechanisms and modeling formalisms for dynamically configurable architectures.

• Different ADLs have varying coverage and support for various kinds of analyses. Several ADLs can be explored for synergies and integrated usage. Some efforts like [65, 80, 107] have already been put forth. There is still a need for further work to improve the overall development process of a product.

• Further exploration into the usage and possible extension of EAST-ADL to support run-time configuration variations. This can include but is not limited to finding the feasibility of using EAST-ADL variability modeling support as a part of DySCAS information model [27].

9.1 Conclusion

The thesis explored different modeling and simulation aspects related to embedded systems in the context of automotive systems. With the aim of enhancing embedded system development processes, a bridge to reduce the gap between specification languages and formal analysis tools is developed. In particular EAST-ADL and timed automata are integrated to enable analysis of architecture specifications. The integration efforts include both conceptual mappings and prototype implementations for validation purpose.

The above mentioned work is focused on one level of abstraction. After ensuring consistency and performing higher level analysis for correctness, the architecture is refined to an implementable form. The thesis has also investigated refinement of EAST-ADL to AUTOSAR from a behavior viewpoint. Overall, the thesis has addressed both horizontal and vertical integration of models.

The thesis also investigated different tools for their support of dynamically configurable systems. Although the existing tools do support many aspects of dynamic configurations, they require extensions and integration in order to completely fulfill the needs related to advanced functionalities.
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


