ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MODEL-BASED TECHNIQUES FOR THE FUTURE INTEGRATION FOR FORMAL VERIFICATION OF CRITICAL REAL-TIME SOFTWARE SYSTEMS

BY
Tiyam Robati

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph. D.

MONTREAL, 19 OCTOBER 2016

ACKNOWLEDGEMENTS

Family is one of nature's masterpieces (George Santayana). I would like to dedicate this thesis to my father, Mohammad Jafar Robati, for supporting me spiritually throughout my life. I am sure he would be very happy and proud of me. I would like to thank my mother and my brother for spending by me through the good and challenging times. Finally, to my lovely son, Diyako, who is the light of my life.

I would like to express my deepest gratitude to my advisor, Dr. Abdelouahed Gherbi, for his guidance, motivation, patience, and for providing me with an excellent atmosphere for my research.

I would like to acknowledge the contribution of Bombardier Aerospace, CMC Electronics and CRIAQ for providing the finance of this study.

TECHNIQUES BASÉES SUR UN MODÈLE POUR L'INTÉGRATION ET LA VÉRIFICATION FORMELLE DE SYSTÈMES LOGICIELS CRITIQUES EN TEMPS RÉEL

Tiyam Robati

RÉSUMÉ

Les architectures avioniques modulaires intégrées combinées avec la norme SAE TTEthernet constituent une infrastructure solide pour le déploiement des applications avioniques a criticités mixtes ayant des exigences strictes en termes de sécurité, fiabilité et de performance. L'intégration de tels systèmes est une tâche d'ingénierie complexe et difficile. Par conséquent, l'approche basée sur les modèles qui offre aux ingénieurs systèmes une méthodologie et les outils de support pour maitriser cette complexité, est d'une grande importance. Dans cette thèse, nous présentons une extension pour le langage de modélisation AADL pour supporter la modélisation des applications avioniques à criticités mixtes déployées sur des architectures IMA basées sur TTEthernet. En particulier, nous présentons un méta-modèle qui étend le méta-modèle de base de AADL avec les concepts et les contraintes de ce domaine. Nous définissons une syntaxe textuelle concrète pour cette extension ainsi que l'implémentation de cette extension en utilisant l'outil OSATE. Par la suite, on construit par-dessus notre extension du langage AADL et nous utilisons les transformations de modèles pour supporter la vérification des modèles de systèmes produits avec cette méthodologie. En particulier, nous proposons une transformation des modèles de systèmes en modèles convenables à la simulation avec DEVs. Finalement, nous illustrons l'approche proposée via une étude de cas fournies par Bombardier, notre partenaire industriel dans le projet. Nous utilisons cette étude de cas pour démontrer notre extension et procéder à la vérification de la contrainte *contention-freedom* d'un ordonnancement TTEthernet.

Mots clés: Système Critique, IMA, TTEthernet, ARINC 653, AFDX, AADL, La Vérification, DEVS

MODEL-BASED TECHNIQUES FOR THE FUTURE INTEGRATION FOR FORMAL VERIFICATION OF CRITICAL REAL-TIME SOFTWARE SYSTEMS

Tiyam Robati

ABSTRACT

Integrated modular avionics architectures combined with the emerging SAE TTEthernet standard provides a strong infrastructure for the deployment of mixed-critical avionic applications that meet stringent safety, reliability and performance requirements. Integrating these systems is a complex and challenging engineering task. Of paramount importance is the development of a model-based approach that can endow system engineers with a methodology and supporting tools to cope with this complexity. In this thesis, we present an extension of AADL, the standard language used for architecture and analysis modeling, in order to enable the modeling of integrated multi-critical avionic applications deployed on TTEthernet-based IMA architectures. To do this, we first present a metamodel for the TTEthernet domain followed by an extension of the core AADL metamodel with concepts and constraints relevant for this domain. In doing so, we define the concrete textual syntax for this extension, and we outline the implementation of this extension using the Open Source AADL Tool Environment (OSATE). To verify the AADL model for TTEthernet, we build on our extension of AADL and leverage model transformations to assess the system models produced with this methodology. In this process, we transform the system models to a target model that is compatible with DEVS formalism in its dedicated simulation environment.

We illustrate the proposed approach using a case study provided by Bombardier, our industrial partner in this project, and we show the benefits of our AADL extension and verification approach to the *contention-freedom* property of the TTEthernet schedule.

Keywords: Critical system, IMA, TTEthernet, ARINC 653, AFDX, AADL, Verification, DEVS

TABLE OF CONTENTS

Page

INTRODUCTION 1
0.1 Problem Statement 3
0.2 Research Objective 3
0.3 Research Contributions 4
0.4 Publication 5
0.5 Thesis Organization 5

CHAPTER 1 BACKGROUND 7
1.1 Time-Triggered Architecture (TTA) 7
1.2 Time-Triggered Protocol (TTP) 7
1.2.1 Time-Triggered Transmission 8
1.2.2 Rate-Constrained Transmission 9
1.2.3 Data Flow integration 9
1.3 Time-Triggered Ethernet (TTEthernet) 11
1.4 Integrated Modular Avionic Architecture (IMA) 15
1.4.1 ARINC 653 17
1.4.2 ARINC664 part7, AFDX 20
1.5 Model-Driven Engineering Approach 21
1.6 The Architecture Analysis Design Language (AADL) 24
1.6.1 AADL Annexes 26
1.6.1.1 Behavioral Annex 27
1.6.1.2 ARINC 653 Annex 27
1.7 Verification and Simulation Techniques 28
1.7.1 Discrete Event System Specification (DEVS) 30

CHAPTER 2 RELATED WORK 33
2.1 TTEthernet-based platform 33
2.2 IMA Architecture 34
2.3 AADL 35
2.4 DEVS 36

CHAPTER 3 TIME-TRIGGERED ETHERNET METAMODEL 37
3.1 TTEthernet Model 37
3.2 A Metamodel for TTEthernet Domain 39
3.2.1 Schedulable Resources 42
3.2.2 Channel 43
3.2.3 Frames 43
3.2.4 Scheduler 44
3.2.5 The Metamodel of IMA 44
3.3 Conclusion 46

CHAPTER 4 AN EXTENSION FOR AADL TO MODEL MIXED-CRITICAL AVIONIC SYSTEMS DEPLOYED ON IMA ARCHITECTURES WITH TTETHERNET 47
4.1 Metamodel Extending AADL capability to model TTEthernet 47
4.2 Textual Syntax for the TTEthernet Extension for AADL 48
4.2.1 Integration of the AADL-TTEthernet Compiler to OSATE2 49
4.3 An Example: A Model of a Subsystem of the Flight Management System 50
4.4 Conclusion 53

CHAPTER 5 VERIFICATION APPROACH TO THE DESIGN OF DISTRIBUTED IMA ARCHITECTURES USING TTETHERNET 55
5.1 Verification Approach 55
5.2 Simulation of the Navigation & Guidance System 61
5.2.1 System Description 61
5.2.2 Model Transformation 62
5.2.3 Property verification: Contention-Freedom 62
5.3 Conclusion 65

CHAPTER 6 CASE STUDY 67
6.1 System Description 67
6.2 Property verification: Contention-Freedom 69

CONCLUSION AND RECOMMENDATIONS 75

BIBLIOGRAPHY 78

LIST OF TABLES

Page

Table 3.1 TTEthernet levels 38

Table 3.2 IMA metamodel in accordance with TTEthernet metamodel.................. 46

Table 4.1 Virtual Links details 52

Table 5.1 Mapping source model into target model 58

Table 6.1 Virtual Links detail.. 69

LIST OF FIGURES

Page

Figure 1.1 TTP example 8

Figure 1.2 TTP example 10

Figure 1.3 Integration Method 10

Figure 1.4 Federated Architecture V.S. IMA Architecture 16

Figure 1.5 An Example of ARINC 653 RTOS 17

Figure 1.6 ARINC 653 module with two partitions 18

Figure 1.7 ARINC 653 hierarchical scheduling example 19

Figure 1.8 Virtual Link 20

Figure 1.9 Virtual Link Flow Regulation 21

Figure 1.10 The Scheduler Flow of Virtual Link 21

Figure 1.11 The Jitter Effect for a Maximum Bandwidth Data Flow 22

Figure 1.12 Model transformation pattern 23

Figure 1.13 ATL overview 24

Figure 1.14 Integration Method 27

Figure 1.15 DEVS a common formalism 29

Figure 3.1 TTEthernet network 38

Figure 3.2 TTEthernet Multi Cluster 39

Figure 3.3 TTEthernet Cluster 40

Figure 3.4 TTEthernet Metamodel 41

Figure 3.5 TTEthernet Metamodel Overview 42

Figure 3.6 Schedulable Resources 43

Figure 3.7 Frame Categories 44

Figure 3.8 TTEthernet Scheduler 45

Figure 3.9 The metamodel of IMA 45

Figure 4.1 AADL-TTEthernet metamodel dependencies 48

Figure 4.2 General structure of a textual syntax framework 49

Figure 4.3 xText grammar overview for AADL-TTEthernet 50

Figure 4.4 AADL-TTEthernet plug-in integrated to OSATE2 51

Figure 4.5 A subsystem of the Flight Management System 51

Figure 4.6 Flight Management Subsystem Model using AADL TTEthernet Extension 53

Figure 5.1 Overall Architecture 56

Figure 5.2 DEVS metamodel 57

Figure 5.3 ATL transformation rules 59

Figure 5.4 Mapping a hierarchical model onto a hierarchical simulator 60

Figure 5.5 The Navigation & Guidance system 62

Figure 5.6 Textual Syntax 63

Figure 5.7 TTEthernet Model 63

Figure 5.8 The corresponding DEVS model 64

Figure 5.9 Simulation graph for the navigation & guidance system 64

Figure 6.1 Case study 68

Figure 6.2 Textual Syntax 70

Figure 6.3 The AADL TTEthernet model 71

Figure 6.4 The DEVS model 72

Figure 6.5 Simulation graph for Case study 73

LIST OF ABREVIATIONS

MDE	Model-Driven Engineering
IMA	Integrated Modular Avionics
TTEthernet	Time Triggered Ethernet
ARINC	Aeronautical Radio, Incorporated
AFDX	Avionics Full-Duplex Switched Ethernet
AADL	Architecture Analysis & Design Language
DEVS	Discrete Event System Specification
SAE	Society of Automotive Engineers
TTE	Time-Triggered Ethernet
RC	Rate-constrained
BE	Best Effort
TTA	Time-Triggered Architecture
TTP	Time-Triggered Protocol
SC	Synchronization Client
SM	Synchronization Master
CM	Compression Master
VL	Virtual Link
FMS	Flight Management System

INTRODUCTION

Avionic systems belong to the class of safety-critical systems that must meet strict safety, reliability and real-time requirements. These systems were designed to used as federated architectures, where each software function is designed and deployed to use exclusive resources. This approach, however, is costly in terms of equipment and wiring. Today, most avionic systems are based on Integrated Modular Avionics (IMA) architectures, where several system functions, each having different levels of safety and performance, might be deployed in the same computing module (e.g. control functions and comfort functions). In fact, IMA architectures are based on an isolation of resources Watkins and Walter (2007), which is achieved through resource sharing between functionalities. IMA-based avionic systems, therefore, demonstrate mixed-criticality and require solid isolation and partitioning. These features are supported in IMA with operating systems and executives compliant with the ARINC 653 standard Aerospace (2011b). IMA architectures are distributed using a communication infrastructure, which can meet the same level of safety and performance requirements.

Ethernet is a widely used standard network (IEEE 802.3) that is not only used as infrastructure for classic office systems but is increasingly used to support industrial and embedded systems due to the high bandwidths it provides. However, Ethernet does not meet strict time constraints for safety critical applications. Several extensions to enhance the predictability of Ethernet have been developed. One of these extensions is the Avionic Full Duplex AFDX standard, ARINC 664 part 7 Incorporated (2009). AFDX is a deterministic real-time extension of Ethernet based on static bandwidth scheduling and control using the concept of virtual links. The Society of Automotive Engineers' (SAE) standard TTEthernet Aerospace (2011d), which was designed to achieve bounded latency and low jitter, is the most recent Ethernet extension based on the time-triggered communication paradigm Kopetz and Bauer (2003) and Obermaisser (2004). A TTEthernet network implements a global time using clock synchronization and offers fault isolation mechanisms to manage channel and node failures. TTEthernet integrates three types of data flow: Time-Triggered (TT) data flow, which is the highest priority flow; Rate Constrained (RC) data flow, also known as AFDX traffic; and Best Effort (BE) data flow. This makes TTEthernet suitable for mixed-criticality applications, such as avionic and automotive applications, where highly critical control functions (e.g. flight management systems) cohabit with less critical functions (e.g. entertainment systems).

IMA architectures with TTEthernet provide a platform to integrate avionic systems and applications with particular features suitable to particular needs. This platform provides error isolation both at the module level, through time and space partitioning, and at the network level, by integrating differentiated data flows. The focus of this research is on avionic applications deployed on IMA architectures interconnected using TTEthernet. The advantages of this infrastructure are numerous (resource sharing, error isolation, integration of data flows). However, these systems are also complex, and the integration of diverse applications with mixed-criticality levels capable of meeting strict real-time constraints is very challenging. In order to control the complexity of such systems, a model-based approach is required which provides system engineers with a methodology and supporting tools to accomplish this integration correctly and efficiently.

A key element of this approach is a modeling language that can allow engineers to express the system at a convenient level of abstraction and to interface with sophisticated formal analysis techniques to verify the safety and performance properties of the system. Architecture Analysis and Design Language (AADL) is a well-established standard modeling language used in the domain of real-time critical systems. AADL has been extended to support the modeling of IMA with an Annex ARINC 653 Aerospace (2011b). However, there is as yet no support for AADL to model the networking of IMA modules through TTEthernet.

0.1 Problem Statement

Mixed-critical applications deployed on distributed architectures are an important issue in many engineering domains. TTEthernet can enrich these architectures and provide an infrastructure with numerous advantages. These advantages are obtained from the combination of distributed architecture (e.g. resource sharing) and TTEthernet (e.g. fault-tolerant). However, the integration of these systems is complex and challenging. To cope with the complex integration of such an infrastructure, we advocate for a model-driven engineering approach (MDE). MDE allows us to produce a methodology that can develop supporting tools that hide the complexity of runtime phenomena from system engineers. Furthermore, using verification techniques, system engineers can ensure that the integration has been accomplished correctly and efficiently. The key element of such an approach is a modeling language that can allow the engineers to express the system at a convenient level of abstraction. The engineers then can set up an interface between the model and tools for sophisticated formal analysis techniques to verify the safety and performance properties of the system. Briefly, the main observations of our research project are summarized as follows:

- There is as yet no modeling framework for IMA architecture interconnected with TTEthernet.
- AADL is a convenient modeling language that has also been used to develop ARINC 653 annex; however, AADL does not support network modeling.
- There is a need for automatic verification of TTEthernet requirements and constraints deployed on distributed systems such as IMA.

0.2 Research Objective

The main objectives of our research are as follows:

a. *Objective 1*: Define and implement an extension for the standard architecture and analysis modeling language AADL to enable the modeling of integrated multi-critical avionic applications deployed on TTEthernet-based IMA architectures.

b. *Objective 2*: Develop a verification approach for the system models produced by our extension of the AADL modeling language.

c. *Objective 3*: Validate our proposed approach using a real-world case study provided by our industrial partner.

0.3 Research Contributions

The main contributions of this thesis consist in the definition of an MDE approach to support system engineers in using TTEthernet. More specifically our contributions involve:

a. A metamodel of the TTEthernet standard; we developed a metamodel to support the SAE TTEthernet standard, AS6802 Aerospace (2011d) for distributed architectures on which safety-critical applications are deployed.

b. An extension for AADL to model mixed-criticality avionic systems deployed on IMA architectures with TTEthernet. This AADL-TTEthernet metamodel describes the structural aspects of a distributed IMA system interconnected using TTEthernet and makes explicit all concepts specified by this standard.

c. A model-based approach to automate the verification of AADL models for extension. We used model transformation techniques to map the AADL-TTEthernet metamodel to the DEVS metamodel in order to simulate the output model using the DEVS simulation environment.

d. Collaboration with SAE AADL committee in developing the AADL networking annex.

0.4 Publication

The main outcome of this thesis in terms of scientific publications are four accepted conference and journal papers and one submitted journal paper. These publications are reported in following:

a. An Extension for AADL to Model Mixed-Criticality Avionic Systems Deployed on IMA architectures with TTEthernet. 1st Workshop on Architecture Centric Virtual Integration@ the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014)

b. Simulation-Based Verification of Avionic Systems Deployed on IMA Architectures. ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS'15).

c. A Modeling and Verification Approach to the Design of Distributed IMA Architectures using TTEthernet. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016). Procedia Computer Science 83, 229-236.

d. Time-Triggered Ethernet Metamodel: Design and Application. Journal of software (JSW Vol. 11, No. 10, October 2016)

e. Design and Simulation of Distributed IMA Architectures using TTEthernet: A Model-Driven Approach. Journal of Ambient Intelligence and Humanized Computing (SI-JAIHC-2016)

0.5 Thesis Organization

This thesis is organized in six chapters. In Chapter 1, we introduce the main concepts and terms that are used in this thesis. In Chapter 2, we succinctly review the most closely related research to our own. Chapter 3 outlines the metamodel of TTEthernet, and this is followed in Chapter 4 by a definition of the AADL extension to support the TTEthernet metamodel. Chapter 5 describes the automatic verification and validation methodology. In Chapter 6, we demonstrates the application of the proposed extension with an illustrative case study. Finally, Conclusion presents our conclusion of the thesis.

CHAPTER 1

BACKGROUND

In order to make this thesis as self-contained as possible, we introduce in this chapter the main concepts of Model-Driven Engineering, TTEthernet, IMA architecture, AADL modeling language and the DEVS simulation environment. These are the main concepts that we are using them in following chapters.

1.1 Time-Triggered Architecture (TTA)

Time-Triggered Architecture (TTA) acts as the computing infrastructure for distributed safety-critical real-time systems. It distributes safety-critical applications into clusters and nodes, and it establishes a fault tolerant global time for the whole system. This global time specifies the communication protocol between clusters and nodes accordingly. It also guarantees time-lines for real-time applications and instant error detection. TTA is based on the Time-Triggered communication protocol (TTP).

1.2 Time-Triggered Protocol (TTP)

TTP has been implemented and used in different domains such as FlexRay in the automotive industry, SAFEbus for serial production and the avionics industry. It is also used for the Airbus A380 and Boeing 787 Kopetz (2003). TTP presents a deterministic, synchronized and congestion-free network based on the IEEE 802.3 Ethernet protocol and is compliant with ARINC 664 part7. Figure 1.1 taken from Domitian Tamas-Selicean and Steiner (2012), provides an example of how this protocol functions.

In this example, there are two *End-System*s, ES_1 and ES_2, and three network *Switch*es, NS_1 to NS_3. Task T_2 on ES_1 sends the TT message m_2 to task T_4, which is mapped on ES_2, while task T_1 on ES_1 sends the RC message m_1 to task T_3 on ES_2. This is assuming that tasks T_1 and T_3 are part of application A_1 and tasks T_2 and T_4 belong to application A_2. Furthermore, A_1 and

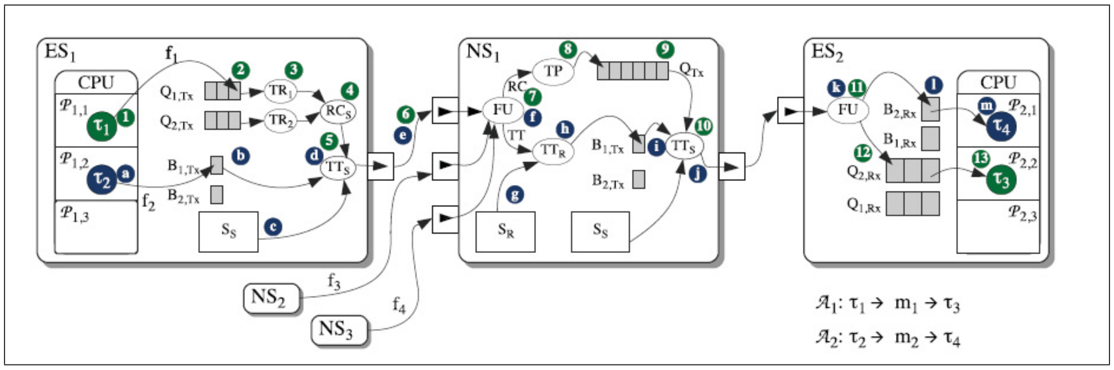


Figure 1.1 TTP example

A_2 introduce different levels of safety-criticality. The isolation of the applications is achieved at the CPU-level through partitioning. Thus, tasks T_1 and T_3 are placed in partitions $P_{1,1}$ and $P_{2,2}$, respectively, while tasks T_2 and T_4 are assigned to partitions $P_{1,2}$ and $P_{2,1}$, respectively. Message m_1 is sent by application A_1 and packed in frame f_1, and m_2 is sent by A_2 and packed into frame f_2. There are two different virtual links, vl_1 and vl_2 (not depicted in the figure), in order to separate the different criticality frames. Frames f_1 and f_2 are transmitted by the switch NS_1, which also forwards frames f_3 and f_4 from NS_2 and NS_3, respectively.

1.2.1 Time-Triggered Transmission

In step (a), task T_2 packs m_2 into frame f_2. Then, in step (b), f_2 is placed into buffer $B_{1.Tx}$ to prepare it for transmission. There is one buffer for every TT message sent from ES_1. There are always static communication schedules that are stored in the form of tables, referred to as routing tables in ESs and NSs. These tables are produced off-line. In this example, the schedule is shown by S. In step (d), f_2, which is a TT task, is sent to NS_1. The duration of this transformation is determined by the schedule and is stored in the S of ES_1 in step (c). Generally, TT tasks are scheduled to be sent before the next scheduled message for transmission. In step (e), f_2 is sent to NS_1 through a dataflow link. The Filtering Unit (FU) checks the integrity and validity of frame f_2 in step (f) and forwards it to the TT receiver task, TT_R, in step (h), which then copies it into sending buffer $B_{1,Tx}$ for later transmission. In the last step, f_2 is sent by the

TT sender task in NS_1 to NS_2. Then, in step (k), f_2 arrives at ES_2, and the FU stores the frame in buffer $B_{2,Rx}$. The task T_4, then, is activated to read f_2 from buffer (m).

1.2.2 Rate-Constrained Transmission

The example of Figure 1.1 shows the Rate-Constrained (RC) transmission, where RC traffic presents event-triggered messages. Frame f_1 is then sent from T_1 on ES_1 to T_3 on ES_2. T_1 packs message m_1 into frame f_1 in step (1) and inserts it into a queue $Q_{1,Tx}$ in step (2). There is one queue per virtual link, where each vl_i carrying an RC frame f_i has a Bandwidth Allocation Gap (BAG). The BAG is calculated and enforced by the Traffic Regulator (TR) task. TR_1 in ES_1 ensures that each BAG_1 interval contains one instance of f_1 shown in step (3). Therefore, each frame leaves the TR task within a specified BAG. The maximum bandwidth used by a vl_i transmitting an RC frame f_i is calculated by the Equation 1.1:

$$BW(vl_i) = f_{i.size}/BAG_i \tag{1.1}$$

An ES can send several messages due to the multiplexing properties of RC messages. Figure 1.2 taken from Domitian Tamas-Selicean and Steiner (2012), demonstrates how the multiplexing of two RC flows coming from TR_i is done. Two RC flows, f_x and f_y, each with a specified size and BAG are illustrated in line (a) and (b). Line (c) shows how the multiplexed flow is performed on the outgoing dataflow link. As demonstrated in line (c), $f_{y,1}$ was delayed with jitter ($f_{y,1.jitter}$) in order to allow for the complete transmission of f_x.

1.2.3 Data Flow integration

This section seeks to describe the mechanisms and consequences of the integration of TT and RC dataflow on the single physical platform. There are three different ways to accomplish this integration: Preemption, Timely Block and Shuffling. As shown in Figure 1.3 taken from Wilfried Steiner and Varadarajan (2009), the purpose of these three integration mechanisms is

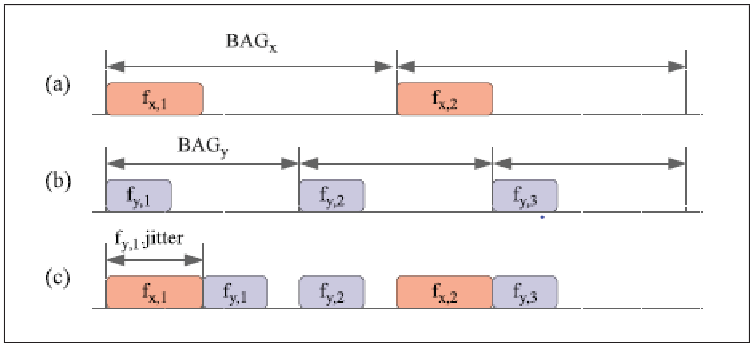


Figure 1.2 TTP example

to clarify when a concurrency exists between two messages with different priorities and what decision should be taken in such a case. If these messages have the same priority, they will be served in FIFO, but in the case of unequal priority, the message with high-priority (H) is served and the message with low-priority (L) will be queued.

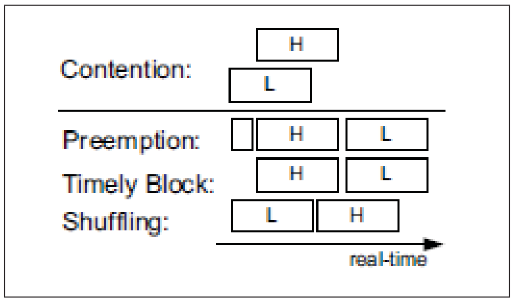


Figure 1.3 Integration Method

Preemption stops the process of relaying message L when message H arrives. The switch takes a minimum of silence time and relays the message H. This mechanism introduces the constant and a priori known latency for message H. But the truncated messages could appear incorrectly to the receiver, which is one of the issues with the preemption mechanism. Two possible solutions to this are first, to include the message length within the message second, to use a signal pattern that violates the line encoding rules when a message is truncated. If a fraction of a truncated message is lost, the whole message will be retransmitted; this causes a loss of bandwidth due to the truncation. Timely Block is a mechanism that ensures switches will not forward messages at those times that TT messages are expected. This mechanism causes more delays while it keeps the scheduled ports free for messages H. The maximum possible number of Ethernet frames for messages L in Timely Block is only 19. To overcome this constraint, the *ES* and *SW* need to act more intelligently; thus when the length of message L is known and transported inside of a given message, the switch will determine if enough bandwidth is available to send message L completely before message H has to be relayed.

Shuffling is an optimal solution that delays message H until the message L process is finished. In the worst case, the delay is equal to the maximum length of message L. This delay also impacts the subsequent message H, because the bandwidth required for the message L is compensated by the sum of the inter-frame gaps between the two succeeding messages H. This mechanism does not truncate a message, nor block the outgoing port for message L, which makes it more efficient then the two previous mechanisms. If message H is a TT message, the real-time quality of the time-triggering is degraded. Latency cannot be mitigated; however, in a 100 Mbit/sec or 1 Gbit/sec network, shuffling still has sufficient real-time quality for applications such as avionics Wilfried Steiner and Varadarajan (2009).

1.3 Time-Triggered Ethernet (TTEthernet)

TTEthernet is a new SAE Standard Aerospace (2011d) that provides time-triggered services for Ethernet in order to allow synchronous communication with constant latency, tight jitter (μ sec) and determinism properties. TTEthernet integrates three data flows: Time-Triggered (TT) data flow, which is the highest priority flow; Rate Constrained (RC) traffic, which is equivalent to AFDX traffic; and Best Effort (BE) traffic. This makes TTEthernet suitable for mixed-critical applications where highly critical functions work alongside less critical functions.

The origins of TTEthernet can be trace back to an collaborative academic project between Vienna University of Technology and TTTech Computertechnik AG TTT. The main objective of this project was to integrate time-triggered messages with event-triggered messages on a single physical Ethernet network.

TT frames are transmitted at specific time intervals established by an off-line time schedule. This schedule specifies the dispatch frame points in time and temporal characteristics for intervals used for asynchronous traffic such as RC and BE. The temporal properties of TT frame f_i are specified by Equation 1.2:

$$f_i[v_x, v_y] = f_i.period, f_i^{[v_x,v_y]}.offset, f_i.length \tag{1.2}$$

The period and length of the TT frame are determined by the off-line configured parameters of the system, and the offset is assigned by the scheduler. The assigned value of the offset for all frames F on all links L in the network is: $F^L.offset$. The dispatch point in time of a TT frame f_i on the communication link $[v_x, v_y]$ is represented by $f_i^{[v_x,v_y]}.dispatch_{pit}$. This is identified by the period and offset of the frame where $f_i^{[v_x,v_y]}$ represents frame f_i transferred as TT on the communication link $[v_x, v_y]$.

The RC traffic, which represents AFDX traffic, guarantees bounded latency in a complex network. AFDX, which is a shorthand for Avionics Full-Duplex Switched Ethernet, ARINC 664 part7 Incorporated (2009), is a predictable communication network that shares network bandwidth between functionalities of a system and maintains the predictability of the communication Brau *et al.* (2013). The characteristics of the RC frame are its maximum transmission rate and length, described by Equation 1.3:

$$f_i = f_i.rate, f_i.length \tag{1.3}$$

The RC frame should always respect its transmission rate limit. In the event that the RC frame exceeds its transmission rate, a traffic policing function implemented in the switch (e.g. leaky bucket) drops the frame. The traffic policing function measures the time between the reception of two frames to determine whether the transmission rate was violated or not.

Finally, BE traffic represents classic Ethernet traffic, where no guarantee exists for the transmission time, reception at the recipient location or delays. In fact, BE frames use the remaining bandwidth of the network due to its lower priority in comparison to TT and RC frames. A typical example of BE traffic is web services.

TT traffic must be free of any conflict. For this to be so, an off-line schedule of frame transmissions that respects the synchronized global time is required. The necessity for there not be any conflict introduces the fundamental constraint of TTEthernet network, known as *contention-freedom*. This definition is formally expressed by Equation 1.4 Steiner (2010), where LCM(F.period) represents the least common multiple of all frame periods known collectively as a *cluster cycle*. It ensures the mutual exclusion of the frames transmitted in the same data flow link, which means that within a given link, only one frame will be transmitted at a certain time.

$$\begin{gathered}
\forall [v_k, v_l] \varepsilon L, \forall f_i, f_j \varepsilon F \\
\forall a \varepsilon [0..(\frac{LCMF.period}{F_i.period} - 1)], \forall b \varepsilon [0..(\frac{LCMF.period}{F_j.period} - 1)] : \\
((F_i \neq F_j) \wedge \exists F_i^{[v_k,v_l]} \wedge \exists F_j^{[v_k,v_l]}) \Rightarrow \\
((a \times f_i.period) + F_i^{[v_k,v_l]}.offset \geq (b \times f_j.period) + F_j^{[v_k,v_l]}.offset + f_j.length) \\
\vee ((b \times f_j.period) + F_j^{[v_k,v_l]}.offset \geq (a \times f_i.period) + F_i^{[v_k,v_l]}.offset + f_i.length))
\end{gathered} \tag{1.4}$$

An off-line schedule established at the time the system was designed is responsible for prohibiting runtime conflicts. Therefore, in the schedule, TT frames have a higher priority than RC and BE frames. When a TT frame and an RC frame arrive in the same outgoing port, the TT frame takes priority over the RC frame. In fact, RC traffic is dispatched if TT traffic is not pending. Therefore, when TT traffic arrives, it will be immediately transmitted. To ensure the immediate transmission, the switch must confirm that the network is free.

TTEthernet is a transparent synchronization protocol that allows different types of traffic to coexist on the same physical communication network. In fact, this synchronization protocol permits transparent integration of time-triggered services on top of standard Ethernet infrastructure.

TTEthernet introduces a fault-tolerant algorithm, which detects failures and disorders in the network. In particular, fault-tolerant algorithms set up the send order for the synchronization messages (i.e. *PCF*s) in order to ensure the synchronization of local clocks in a distributed system. *Protocol Control Frame (PCF)* is a dedicated Ethernet frame, which carries the TTEthernet protocol control frame to synchronize the local clock. *Protocol Control Frame* includes the transmission of overhead protocols from higher layer protocols such as IP and UDP. Therefore, a multitude of TTEthernet End-Systems generate *PCF*s and distribute them with TTEthernet switches. Fault-tolerant algorithms use multiple redundant paths established by the TTEthernet network in order to tolerate the failure of a single path without affecting the entire system applications. It is vital for safety-critical systems to have fault-tolerant algorithms. Multiple redundant paths in the system ensure that even multiple faults can be tolerated.

As previously mentioned, TTEthernet provides local clock synchronization in distributed systems. To do so, a synchronization approach must be established. The main elements of this synchronization approach include a *Synchronization Master (SM)*, a *Compression Master (CM)* and a *Synchronization Client (SC)*. Based on the requirements on the system architecture, either *SM* or *CM* should be selected. Once the system designer decides on the configuration of *SM* and *CM*, the remaining components are configured to be the *SC*. The synchronization approach of TTEthernet is organized in two steps. In the first step, *SM*s send *PCF*s to the *CM*s. Then, after a new calculation, a new *PCF* is sent out from the *CM*s to the *SM*s and, in the second step, to the *SC*s. The new *PCF* contains an average value of arrival times of dispatched *PCF*s in the first step.

The synchronization topology is configured at different levels in the system architecture. The lowest level of this topology is composed of ESs and switches, which are configured as *SM*, *CM* and *SC*. The next level presents the concept of *cluster*, where a single *synchronization domain* and *synchronization priority* are considered. TTEthernet introduces different *synchronization domains* and *synchronization priorities* in order to support system-of-system communication. *Synchronization domains* refer to independent TTEthernet systems inside of a system-of system that respects their *synchronization priorities*. It is important to point out that two components belonging to different synchronization domains will never synchronize their local clocks. That means the communication between two components of different synchronization domains is only possible with non-time-triggered traffic classes. The concept of *cluster* is defined in TTEthernet to permit the running of different *cluster*s in a large TTEthernet network in isolation. A cluster is organized as a set of ESs and switches that are connected using communication redundance channels. These communication channels contain at least one switch. Several clusters constitute a *multi cluster* in the synchronization topology where one *synchronization domain* and many *synchronization priorities* are introduced. A *multi cluster* system supports a master-slave paradigm, which tries to synchronize all devices in the system while respecting the highest synchronization priorities. Finally, the *network* level of the synchronization topology is composed of several *multi clusters* with different *synchronization domains* and *synchronization priorities*.

1.4 Integrated Modular Avionic Architecture (IMA)

The main idea underlying the concept of IMA architecture is the sharing of resources between some functions while ensuring their isolation to prevent any interference Lauer (2010b, 2013); Michaël Lafaye and Pautet (2010); A. Al Sheikh (2010); Watkins and Walter (2007). This contrasts with the federated architectures, where each function is designed and deployed to use exclusive resources. Avionic systems are now based on IMA architectures, where several system functions with different safety and performance requirements might be deployed on the same computing module Watkins and Walter (2007). Figure 1.4 taken from Inc. (2008), demonstrates the difference between federated and IMA architectures for cockpit display, Air data and Flight Management System (FMS) functionalities. In federated architectures, the functionalities are implemented on separate processing units, and these processing units are connected using ARINC 429 for a network. But in IMA architecture, all functionalities are deployed on the same processing unit and managed with an operating system. The resource sharing in IMA architecture reduces the cost of voluminous wiring and equipment, while the non interference guarantee is required for safety reasons.

The IMA architecture is defined by the ARINC 653 standard Incorporated (2013). Each functionality in the system is implemented by one or a set of functions distributed across different modules. A module represents a computing resource hosting many functions. Functions deployed on the same module may have different criticality levels. For safety reasons, the functions must be strictly isolated using partitions. IMA-based avionic systems, therefore, have a mixed-criticality that requires solid isolation and partitioning. These features are supported in IMA with operating systems and executives compliant with the ARINC 653 standard Incorporated (2013). The partitioning of functionalities is done in two dimensions: spatial partitioning and temporal partitioning. The spatial partitioning is implemented by statically assigning all of the resources to the partition executed in a module, where no other partition can have access to the same resources at the same time. The temporal partitioning is implemented by allocating a periodic time window dedicated to the execution of each partition.

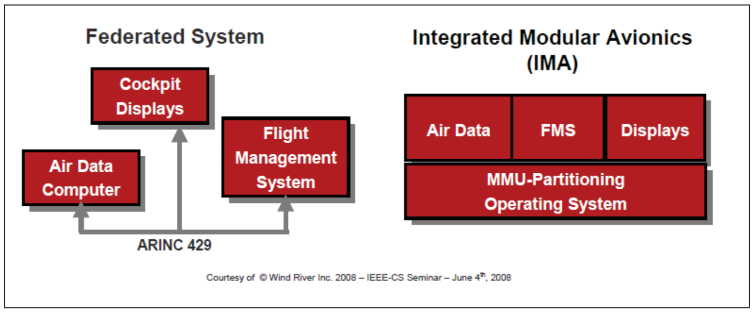

Federated System
Cockpit Displays
Flight Management System
Air Data Computer
ARINC 429
Integrated Modular Avionics (IMA)
Air Data
FMS
Displays
MMU-Partitioning Operating System
Courtesy of © Wind River Inc. 2008 – IEEE-CS Seminar – June 4th, 2008

Figure 1.4 Federated Architecture V.S. IMA Architecture

The segregation and partitioning in IMA are accomplished with the ARINC 653 Real Time Operating System (RTOS). ARINC 653 Incorporated (2013) manages the computational resources of IMA and performs temporal and spatial isolation between partitions. Figure 1.5 taken from Inc. (2008), presents an example of the ARINC 653 RTOS composed of one module hosting four partitions deployed on a hardware board using the ARINC 653 Application Executive.

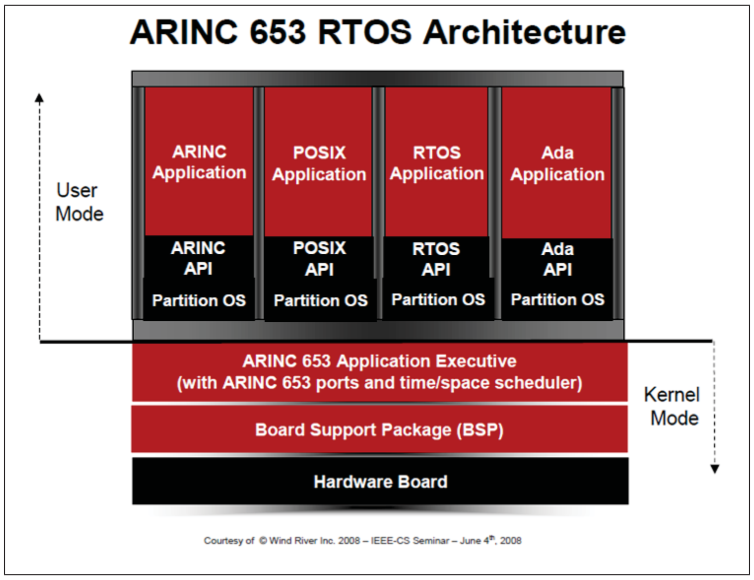

ARINC 653 RTOS Architecture
User Mode
ARINC Application
POSIX Application
RTOS Application
Ada Application
ARINC API
POSIX API
RTOS API
Ada API
Partition OS
Partition OS
Partition OS
Partition OS
ARINC 653 Application Executive (with ARINC 653 ports and time/space scheduler)
Board Support Package (BSP)
Hardware Board
Kernel Mode
Courtesy of © Wind River Inc. 2008 – IEEE-CS Seminar – June 4th, 2008

Figure 1.5 An Example of ARINC 653 RTOS

Two ARINC standards define IMA systems, ARINC 653 Incorporated (2013) and ARINC 664 Incorporated (2009). In following sections, we briefly explain them.

1.4.1 ARINC 653

ARINC 653 is a real-time operating system produced by ARINC Corporation. Isolation between partitions, which ARINC 653 accomplishes is particulary important because failure in a partition should not affect the functionality of other partitions that run on the same processor (module). Furthermore, partitions demand strict access to processing resources and memory shared between them. Therefore, the need for temporal and spatial isolation between partitions on the same module is obvious. In temporal partitioning, each partition is executed in a dedicated time slot defined at system start-up. The spatial partitioning dedicates a predetermined amount of memory space, which is also determined at system start-up Brau *et al.* (2013).

Figure 1.6 Julien Delange and Kordon (2009) presents an example of the conceptual model behind ARINC 653: a system composed of two partitions with different criticality levels, where partition 1 has a higher level of criticality. The communication between them is realized by the ARINC 653 kernel using one communication channel from partition 1 to partition 2, which allows unidirectional data transmission from partition 1 to partition 2. The ARINC 653 module is responsible for managing the address space in memory in order to isolate the partition code and data, and it also manages the time slot to execute partitions.

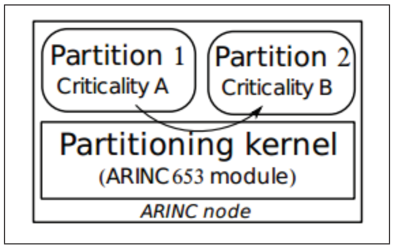


Figure 1.6 ARINC 653 module with two partitions

The concept of hierarchical scheduling is performed in ARINC 653 in two levels; kernel or module level, partition level, illustrated in Figure 1.7 taken fromJulien Delange and Kordon (2009). The kernel level is a static scheduling and executes each partition cyclically at a given rate. The scheduling policing which is defined by system designer is performed in the partition level. That enables defining different scheduling policy per partition.

ARINC 653 modules realize two types of communication, communication between ARINC 653 processes in the same partition (intra-partition communication) and communication between ARINC 653 processes across partitions (inter-partition communication). In the case of interface communication between ARINC 653 processes in the same partition, no kernel or module is used. Therefore, they are isolated because failure of an intra-partition communica-

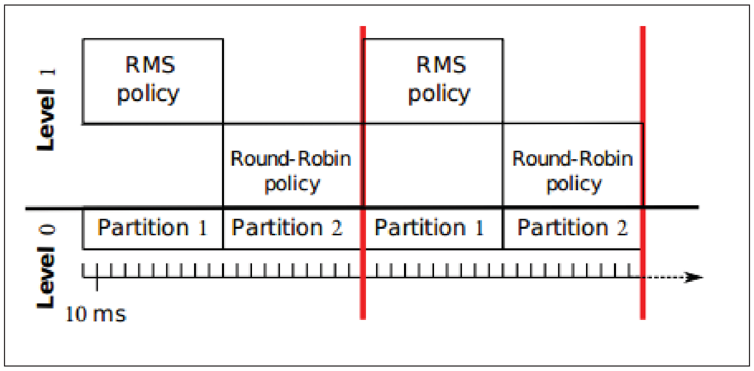


Figure 1.7 ARINC 653 hierarchical scheduling example

tion does not cause failure in other partition functionalities. The intra-partition communication is established by four mechanisms, as listed below:

- The buffer stores multiple messages in the message queue (FIFO, Priority).
- The blackboard stores one instance of a message until it is cleared or overwritten by a new instance.
- The event notifies the completion of a job.
- The semaphore controls access to shared resources.

The inter-partition communication is supervised by the module. The ports' routing policy is statically defined by the system designer. The inter-partition communication proceeds as follows:

- The queuing ports store multiple messages in queues.
- The sampling ports are similar to blackboard services in intra-partition communication.

During the design and development process of an ARINC 653 system, several issues should be taken into account to address needs of reliability and robustness (e.g. partition scheduling, resources dimensioning). All scheduling policies must be validated in order to ensure there is enough time for their execution. All resource dimensions must be corrected in accordance with run time requirements. Therefore, no unexpected deadlock or crash should happen. For example, the buffer size should be checked to avoid buffer overflow at execution time. The validation of these requirements needs a lot of testing and implementation efforts. However, they can be validated at the design-level before any implementation has begun, which can reduce the need for testing efforts and error detection.

1.4.2 ARINC664 part7, AFDX

One established communications medium in IMA is Avionics Full DupleX Ethernet (AFDX) Incorporated (2009), which is a deterministic real-time network based on Ethernet. AFDX supports Rate Constrained (RC) traffic, which is event-triggered traffic, and uses the concept of Virtual Link (VL) in order to share bandwidth between partitions and modules of IMA architecture. VL is a unidirectional logical connection from one sender module to one or more receiver modules. This is shown in Figure1.8 taken from Incorporated (2009).

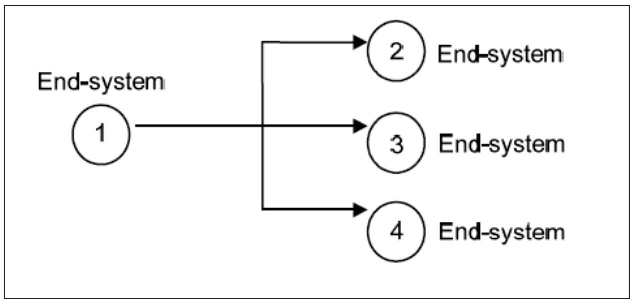


Figure 1.8 Virtual Link

VL replaces point-to-point cabling used in federated architectures and uses traffic shaping to regulate the time between two consecutive frames that would be sent on the same VL. This leads to bandwidth controlling that provides traffic at a constant and deterministic rate Ahmad Al Sheikh and Hladik (2013). Figure 1.9 Incorporated (2009) shows this regulated flow for a single VL. VL has two main characteristics: Bandwidth Allocation Gap (BAG) and Maximum Frame Size (MFS). BAG is defined as the minimum time interval between two consecutive frames on an AFDX network, and MFS is the maximum size of a transmitted frame on a VL. This is shown in Figure 1.9 Incorporated (2009).

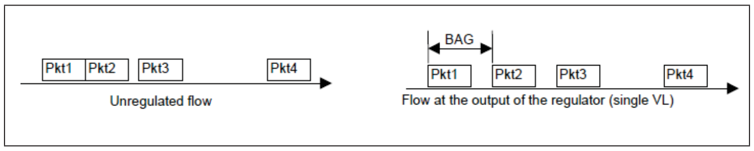


Figure 1.9 Virtual Link Flow Regulation

In the case of a transmission using multiple VLs, the scheduler multiplexer is used to manage the multiplexing of different flows coming from the regulator, as depicted in Figure 1.10 Incorporated (2009).

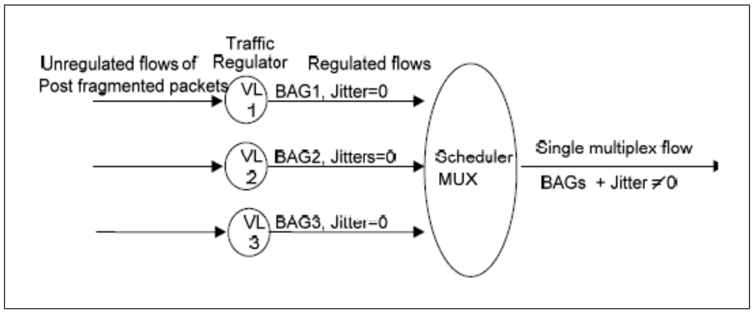


Figure 1.10 The Scheduler Flow of Virtual Link

At the output of the scheduler multiplexer, frames appear in bounded time intervals, as in Figure 1.11 Incorporated (2009), where the maximum possible jitter is respected. This jitter is produced by the scheduler, not by the traffic flow in the AFDX network.

1.5 Model-Driven Engineering Approach

In the domain of software engineering, there is often a wide conceptual gap between a problem and the implementation of an effective solution. The bridging of this gap is usually done by a systematic transformation of a real problem into an implementation domain, where the real

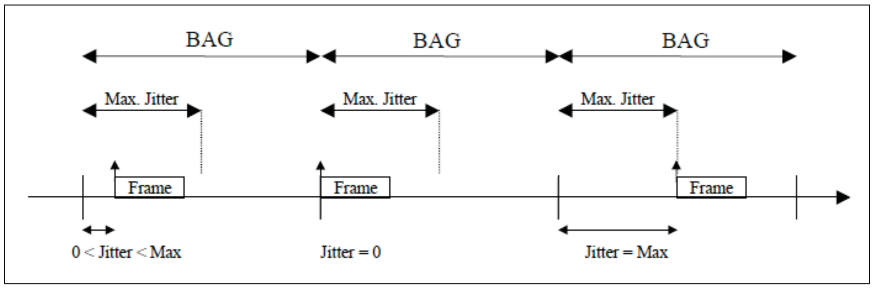


Figure 1.11 The Jitter Effect for a Maximum Bandwidth Data Flow

problem is represented by a model at multiple levels of abstraction, France and Rumpe (2007). This process is known as MDE.

Model, *metamodel* and *metametamodel* are key elements of MDE. A *model* represents a system that refers to the real-world. This representation contains the characteristics of the system and any knowledge about it. A *metamodel* defines a languages that enables the expression of models. Moreover, it describes the elements of a model, the relation between these, as well as the constraints that should be respected by the model. A *metamodel* defines the abstract syntax of modeling languages F. Jouault and Kurtev (2008). The conceptual foundation of a *metamodel* is captured in a model called a *metametamodel*. Figure 1.12 shows the common pattern for model transformations in MDE. M1, M2 and M3 are three levels of abstraction of this architecture representing *model*, *metamodel* and *metametamodel*. Globally, a *model* is defined in conformance with a *metamodel*, and a *metamodel* in conformance with a *metametamodel*. In Figure 1.12 taken from F. Jouault and Kurtev (2008), Tab represents a transformation language, such as ATL, which is responsible for the automatic generation of *Mb* by executing *Ma*, where *Ma*, *Mb* and Tab conform to *MMa, MMb* and *MMt*, respectively. All three metamodels conform to the *metametamodel MMM*, which could be MOF or EMF. In our context, *MMM* is EMF.

Eclipse is an open universal tool platform for software development, particularly for the construction of IDEs (integrated development environments). The Eclipse Modeling Framework (EMF) is the most used environment for MDE. It provides an underlying modeling language

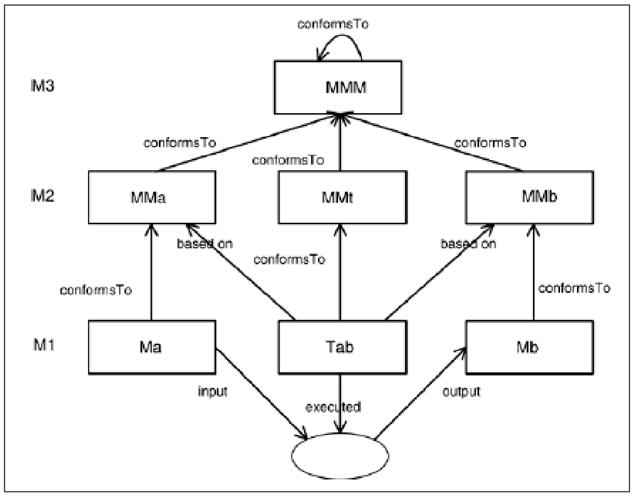


Figure 1.12 Model transformation pattern

called *Ecore* as well as a code generation framework. The adaptable extensibility of EMF provides a solid foundation for many model-based language development tools. In fact, EMF supports creating, modifying, storing and loading instances of models by describing class models and Java code generation. It incorporates JAVA, XML and UML. An EMF model is the common representation of these languages regardless of what technology the utilized technology to define a model Biermann *et al.* (2006).

The metamodel of EMF, which is also called EMF core model, contains elements such as *EClass*, *EDataType*, *EAttribute* and *EReference*. *EPackage* arranges *EClasse*s in order to perform sub-packages, the elements of model as well as the relation between these. Also, the EMF metamodel contains some abstract classes, such as *ENamedElement*, *ETypedElement* and others, to help better structure the model.

The ATLAS Transformation Language (ATL) F. Jouault and Kurtev (2008) is a domain-specific language for specifying model-to-model transformations. An ATL transformation program is composed of transformation rules that define how source model elements are mapped into the elements of target models. Figure 1.13 taken from F. Jouault and Kurtev (2008), presents the transformation pattern of the ATL model transformation language. In this Figure, *Ma* is the source model that is transformed into *Mb*, the target model, according to the ATL transformation rules written in *mma2mmb.atl*. The model representing the ATL transformation definition conform to the ATL *metamodel*. At a higher level, all metamodels conform to EMF. The Eclipse environment contains a set of tools and features that have been adapted and extended to best suit the needs of ATL development.

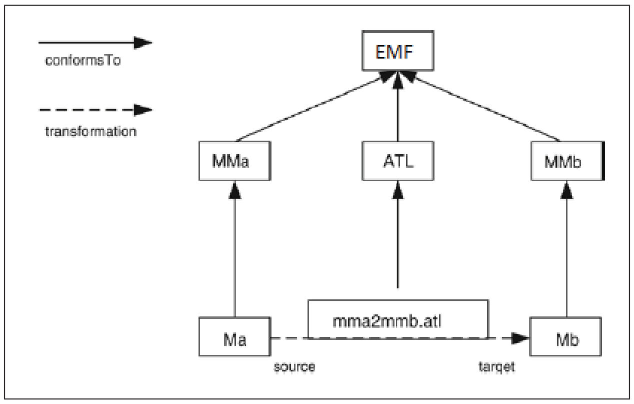


Figure 1.13 ATL overview

1.6 The Architecture Analysis Design Language (AADL)

Many model-based languages exist, such as Architecture Analysis Design Language (AADL), Unified Modeling Language (UML), System Modeling Language (SysML), Analysis of Real Time and Embedded systems (MARTE), and others. Of these languages, AADL is the most appropriate for our project due to its extensibility properties and its already developed and published annexes, which include ARINC 653Incorporated (2013), behavioral annex Aerospace (2011c) and error modeling annex Aerospace (2011a). Moreover, AADL is an open source software built on the Eclipse Modeling Framework. Our proposed tooling set also builds on the Eclipse ecosystem, and our motivation for choosing AADL was mainly informed by its interoperability. The AADL SAE group (AS-2D) also helps us a lot in terms of presenting our progress every three months and conducting our project, which will be published as the next AADL annex called *Networking Annex*. The following section will present more details about these modeling languages.

AADL is a standard architecture description language developed by SAE AS5506 SAE (2012) for formal specifications of hardware and software architectures of embedded computer systems. It focuses on the distinct components and the interaction between components Pi (2009). It also describes the dynamic architecture of an embedded system, the constraints of a real-time system and the mapping of software to hardware components Frana (2007). AADL is used for the modeling of software system architectures and supports the analysis and verification of non-functional properties of modeled system (i.e. quality attributes). More specifically, AADL is used to model software system architectures and its deployment on the execution platform. A number of operating system characteristics, including communication and synchronization mechanisms and thread behavior, are directly supported by the language.

In the AADL execution model, both synchronous and asynchronous aspects are mixed Frana (2007). A synchronous execution model is defined by logically synchronized periodic threads communicating through data ports. The value transmission from output port to input port is done at the beginning of the period. Two threads in the same period can communicate together by means of the existing immediate transfer protocol, which implements a zero-time computation hypothesis. To validate the synchronous hypothesis, the real-time properties attached to model elements is used. AADL also introduces an asynchronous model, which allows it to declare buffered data, to raise events on events or an event of the data port, to specify sporadic and aperiodic threads with different periods, which can communicate together through shared variables and remote procedure calls. The execution model is based on automatic modeling stopwatches. For that, the execution time of threads is defined and compared with the requirements deadline.

AADL provides different types of components with precise semantics in order to express the entire system. AADL components that represent the elements of architecture are listed as follows:

- Software components such as *thread, thread group, subprogram, data and process*.

- Hardware components, which include *processor or virtual processor, memory, bus/virtual bus and device*.
- Hybrid components such as *system*, which is used to describe the hierarchical grouping of hardware and software components.

Thread, which is the only schedulable component of AADL, represents a sequential flow of execution. Different *thread*s communicate together through *data flow*s. *Subprogram* represents the piece of code that can be called on by a thread or another subprogram. The connection points are defined by the interface of the communication components such as *ports, data accessing*. A *Component* is defined by a *type* in order to define the component's external interface, and its *implementation* to define the internal structure of the component. The management of large and complex systems is performed in AADL by using *Package*s, which define name space and component libraries.

AADL is an extensible modeling language that uses two extensibility mechanisms. The first mechanism is a construct for property set definition. This construct enables the defining or modifying of AADL properties. The second extensibility mechanism is an annex extension mechanism, which enables it to specify sub-languages that will be processed within an AADL model. Some AADL annexes are now standardized, such as the Error Modeling Annex Aerospace (2011a), which allows the specification of error models to be associated with core components supporting safety and dependability modeling.

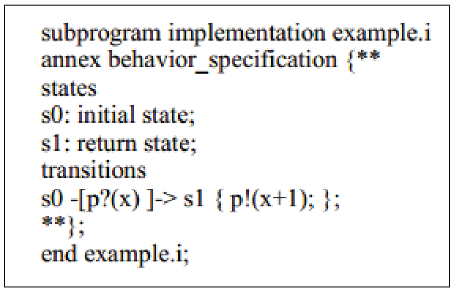
An open source tool set built on Eclipse plug-in technology is provided for AADL. Known as OSATE CMU/SEI (2014), it is implemented by the Society Automotive Engineers (SAE) standard AADL Liu and Gluch (2009).

1.6.1 AADL Annexes

As mentioned above, AADL is an extensible modeling language that uses two extensibility mechanisms, property set definition and the annex extension mechanism. Some AADL annexes have already been standardized and published such as ARINC653 and Behavioral annexes. In this thesis, we do not describe these annexes in detail, but in the next section we briefly review them in order to explain the concept of developing a new annex in AADL.

1.6.1.1 Behavioral Annex

The behavioral annex of AADL (AADL-BA) Aerospace (2011c) provides constructions to define the expected behaviors of system components described by AADL. It is an automata-based annex. This annex is used to describe the behaviors of port communication, subprogram call, timing, and others. Because it is an extension of the dispatch mechanism of the execution model, the role of the AADL execution model is to determine when a behavior annex is processed and what data it executes. The behavioral annex can be attached to the thread or subprogram, as shown in Figure 1.14 taken from Pi (2009). In this example, the system has two states (initial state and return state) and a transition state between them.



```
subprogram implementation example.i
annex behavior_specification {**
states
s0: initial state;
s1: return state;
transitions
s0 -[p?(x) ]-> s1 { p!(x+1); };
**};
end example.i;
```

Figure 1.14 Integration Method

1.6.1.2 ARINC 653 Annex

In the previous section, we explained the functionalities of ARINC 653 in IMA architecture. In this section, we focus on modeling ARINC 653 using AADL. In order to provide safety-critical services in this model, space and time partitioning of ARINC 653 avionics standard is necessary. The first version of AADL was not amenable to the present model for ARINC 653, particularly in the matter of isolation requirements for the system. This was the main reason for developing a new annex in order to support the ARINC653 specification.

An ARINC653 module is modeled by a processor component Julien Delange and Kordon (2009). This processor models partitioning functionalities. Therefore, it contains partitions runtime as a sub-component, and it defines partitions scheduling policy as component properties. Virtual processor models partition runtime such as scheduling policy for partition tasks, partition resources and so on. The process component models the partition address space and contains thread and data, which are the partition content. The AADL property *Actual-Processor-Binding* combines the virtual processor and process together. Also *Actual-Memory-Binding* combines memory with a process component in order to facilitate the allocation of memory segments.

1.7 Verification and Simulation Techniques

Simulation is defined as providing the model of a real system or a real problem in order to undertake experiments toward describing the behavior of that system or problem. This process concludes by evaluating various scenarios for the system. In fact, the output result obtained from the simulation is used for making better decision during the implementation process of most engineering projects, which contributes to better efficiency, system performance, and error detection in early stages of design. Two different types of simulation are reported by Klingstam and Gullander (1999):

- Discrete event simulation (DES)
- Geometric simulation (GS)

DES simulates the behavior of a system at a discrete point in time, whereas GS simulates continuous time. In this project, we mostly focus on DES due to the nature of TTEthernet, which includes time-triggered and event-triggered messages.

Figure 1.15 illustrates a Formalism Graph Transformation (FGT) Vangheluwe (2000). The different formalisms are the nodes of this graph, and the solid arrows represent mapping relations between formalisms. The dotted, vertical thick arrows indicate the existence of a simulator that can map an abstract model onto a state trajectory. The dashed line in the middle distinguishes the crude division between continuous and discrete formalisms. As we can see from Figure 1.15 taken from Vangheluwe (2000), DEVS is a common denominator for the representation of discrete-event and continuous-time models. This is because, firstly, DEVS is the most appropriate and most common formalism in targeting simulation. Secondly, the semantics of discrete-event formalisms, such as Event Scheduling, Activity Scanning and Process Interaction, can be declared using DEVS.

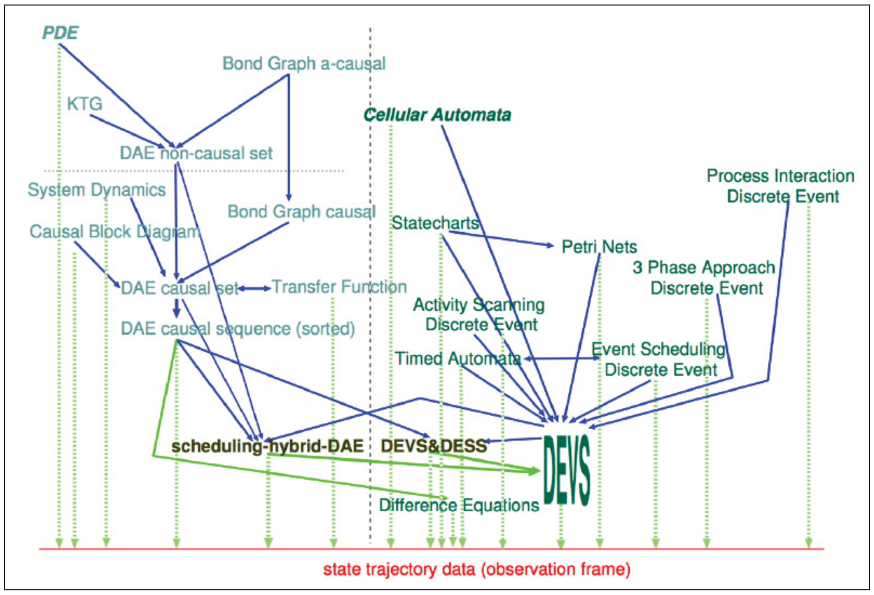


Figure 1.15 DEVS a common formalism

DEVS is a super-formalism that surrounds the expressiveness of the individual formalism. The super-formalism is an alternative to analyzing complex multi-formalism systems. Multi-formalism systems are a modeling approach for complex systems where no single analysis or modeling method can successfully tackle all aspect of the systems. Another benefit of DEVS is its high level syntactic elements, which enables learning about the semantics of the high-level formalism by transforming from high-level to low-level. In our context, the rich syntactic elements in the AADL modeling language are modeled explicitly using DEVS. The property required to describe the AADL model with DEVS increases as a result of transformation.

1.7.1 Discrete Event System Specification (DEVS)

DEVS formalism, Zeigler. (1984) provides a rigorous common basis for discrete-event and continuous-time modeling and simulation. It is presented as an extension to Finite State Automata that describes the behavior of systems in two levels, *atomic* DEVS and *coupled* DEVS. The behavior of a discrete-event system is described with the help of *atomic* DEVS, which takes advantage of Finite State Automata to produce output events from the reaction to input event. *atomic* DEVS is structured using Equation 1.5, Zeigler. (1984). T is the continuous time base, state set of S is the set of admissible sequential states. Concurrent parts of a system are defined by n. The time advance function, ta, is used to model the time in the system. The internal transition function, δ_{int}, describes the behaviour of a Finite State Automaton. The output set, Y, denotes the set of admissible outputs, whereas ℓ represents the output ports of systems. The output function, λ, is responsible for mapping the internal state onto the output set. Q denotes the total state of the system, whereas e refers to elapsed time. σ describes the time left in a state. In the case that the system receives an external set of inputs, X represents all admissible input values and m for input ports. The set Ω contains all admissible input segments ω. Finally, the reaction of systems into all external events is represented by δ_{ext}.

$$atomicDEVS \equiv \langle S, ta, \delta_{int}, X, \delta_{ext}, Y, \lambda \rangle \tag{1.5}$$

Where,

$$\begin{gathered}
T = R \\
S = \times_{i=1}^{\ell} Y_i \\
t_a : S \rightarrow R_{0,+\infty}^{+} \\
\delta_{int} : S \rightarrow S \\
Y = \times_{i=1}^{\ell} S_i \\
\lambda : S \rightarrow Y \bigcup \{\phi\} \\
Q = \{(s,e)|s\varepsilon S, 0 \leq e \leq ta(s)\} \\
\sigma = ta(s) - e \\
X = \times_{i=1}^{m} X_i \\
\omega : T \rightarrow X \bigcup \{\phi\} \\
\delta_{ext} : Q \times X \rightarrow S
\end{gathered} \tag{1.6}$$

A *coupled* DEVS represents the overall system as a network of coupled components. These components can be *atomic* DEVS or *coupled* DEVS in their own right Zeigler. (1984). *coupled* DEVS is structured using Equation 1.7, Zeigler. (1984):

$$coupledDEVS \equiv \langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}, select\rangle \tag{1.7}$$

Where,

$$
\begin{gathered}
\{M_i | i \varepsilon D\} \\
M_i = \langle S_i, ta_i, \sigma_{int,i}, X_i, \sigma_{ext,i}, Y_i, \lambda_i \rangle, \forall i \varepsilon D \\
\{I_i | i \varepsilon D \bigcup \{self\} \langle \\
\forall i \varepsilon D \bigcup \{self\} : I_i \\
\{Z_{i,j} | i \varepsilon D \bigcup \{self\} : I_j\} \\
select : 2^D \to D \\
select(E) \varepsilon E
\end{gathered}
\tag{1.8}
$$

Self represents the coupled model, X_{self} and Y_{self} denotes the set of allowed external inputs and outputs to the coupled model. The set of unique component references is represented by D. M_i is a set of components that are *atomic* DEVS. The coupling network is denoted by I_i. $Z_{i,j}$ describes how the output of a component is mapped to the input of other components. In fact, Z is a translation functions, where different components such as atomics or coupled influence themselves. Finally, a *select* function is used for tie-breaking between simultaneous events in a *coupled* DEVS.

CHAPTER 2

RELATED WORK

In this chapter, we present an overview of the most closely related research work to our own. The chapter is organized in two parts. In the first, we review research work that focuses on issues and techniques related to TTEthernet-based distributed architectures, such as IMA architectures. In the second part, we review research work in the domain of AADL. Finally, we discuss research work in the domain of the DEVS simulation environment.

2.1 TTEthernet-based platform

Several research works Kopetz (2003), Abeni and Buttazzo (1998) have focused on real-time systems with mixed-criticality requirements. They describe how Time-Triggered (TT) tasks and Event-Triggered (ET) tasks can be integrated in the same physical platform in order to support mixed-critical applications. Izosimove et al. Izosimov *et al.* (2008) presented TT and ET tasks that share the same processor, and they also addressed the problem of mapping and partitioning. The order of tasks (TT and ET) decided by different scheduling approaches is reported by a number of researchers, but this is outside the scope of our work. Braun et al. Braun *et al.* (2001) included task mapping to heterogeneous architectures.

Steinhammer et al. Steinhammer (2007) described and implemented a prototypical TTEthernet controller in FPGA. Many approaches have attempted to adapt Ethernet technology for deployment in applications that require temporal guarantees.

Modeling TTEthernet has been mainly used for simulation purposes. Steinbach et al. Steinbach *et al.* (2011) developed an extension for the OMNeT++ INET framework to support the simulation of TTEthernet. Zhang et al. Zhang and Koutsoukos (2013) introduced and developed a TTEthernet model using SystemC/TLM in order to facilitate the design and integration of a Cyber Physical System (CPS). They integrated the TT task, and they proposed to integrate the TT and ET tasks together in future work, although the efficiency of the simulation in their work meets TTEthernet requirements. Abuteir et al. Abuteir and Obermaisser (2013) introduced a TTEthernet simulation environment based on OPNET for generic building blocks such as switches and systems.

2.2 IMA Architecture

Michaël Lafaye and Pautet (2010) and Lafaye (2010) are two works of the same group that have proposed a modeling approach that describes different levels of detail the IMA execution platform. They propose a modeling approach that computes worst case traversal time (WCTT) for IMA architecture interconnected with AFDX. They produced a functional analysis using a model-checking verification approach. This work fits with the early phase of the development process. They used two standard languages: AADL, to model the high level abstraction of IMA platform; and SystemC, to refine the description of the architectural platform and to provide simulation results. The difference between our project and this work is that, when we target the TTEthernet, the communication network inside IMA remains AFDX. Moreover, they used SystemC languages to refine the AADL model and simulation. We only use the AADL modeling language.

Lauer (2010a) proposed a modeling approach for IMA platforms based on time-automation (as a formal modeling approach). Again, they computed the worst case traversal time (WCTT) for the AFDX network in IMA architecture and produced a functional analysis using a model-checking verification approach. The communication network of the IMA platform is always AFDX. In terms of verification, they used a model-checking approach, which is also interesting for us. In a similar vein, Lauer (2011b) and Lauer (2011a) continued to work on worst case temporal consistency and latency and freshness analysis for IMA platform with different evaluation methods, such as the tagged signal model and Integer Linear Programming (ILP).

A. Al Sheikh (2010) presented an integer linear programming formulation for resource scheduling in IMA architecture that takes resource and temporal constraints into account. This work is interesting to us because it recognizes the main requirements and constraints of the IMA platform that should be verified. Two next work of this group, Ahmad Al Sheikh and Prabhu (2012) and A. Al Sheikh (2009) focus on scheduling and task mapping in IMA architecture.

Lauer (2013) focused on IMA architecture for TTEthernet in order to present a cost effective strategy for integrating multi-critical functions in IMA architecture. This work also used a binary integer problem formalization with an off-the-self solver.

2.3 AADL

AADL presents two extension mechanisms, namely property sets and sub-languages (i.e. annex). Several AADL extensions based on these mechanisms are now standardized as official annexes. These include the Data modeling annex, ARINC653 annex, AADL Behavior Annex Aerospace (2011b), and Error Model Annex Aerospace (2011a). Some works have focused on extending the language using these extension mechanisms or in other ways.

The closest research work to our own is reported in Julien Delange and Kordon (2009) and Lasnier (2011). Delange et al. Julien Delange and Kordon (2009) presented an approach based on AADL, which covers the modeling, verification and implementation of ARINC653 systems. The authors described the modeling guidelines elaborated in the ARINC653 annex of the AADL standard. This approach is supported by a tool chain composed of the Ocarina AADL tool suite ISAE, AADL/ARINC653 runtime POK Dolange and the Cheddar scheduling tool LYSIC Team. Lasnier et al. Lasnier (2011) present an implementation of the AADL behavior annex as an extension plug-in to OSATE 2. We implemented our AADL TTEthernet extension using similar techniques. Michaël Lafaye and Pautet (2010) defined a modeling approach based on AADL and SystemC, which focuses on the design and dynamic simulation of IMA-based avionics platform. This is a component-based approach, which can be used to dimension the architecture taking into consideration the application to be deployed while achieving early platform validation. De Niz and Feiler (2007) discussed how to extend the AADL language to include new features for the separation of concerns (i.e. Aspects). Based on this work, it seems that the AADL extension mechanisms do not support the separation of concerns, and new aspect-like constructs and mechanisms are being investigated. Brau et al. Brau *et al.* (2013) present a model for the subsystem of a Flight Management System using AADL, and they show how to establish important parameters in the AADL model, including, for instance, virtual link characteristics.

Pi (2009) and Frana (2007) are other works that have explained the behavioral annex road map in detail. Liu and Gluch (2009) discussed the different formal model-checking tools used for the verification of AADL behavioral models, such as UPPAAL and NuSMV.

2.4 DEVS

In this section, we briefly outline the most relevant related research with a focus on the model transformation approach, which is used to support verification and simulation. In some M. (2007), models using UML state charts were transformed into DEVS model to overcome the gap between the UML graphical modeling elements and DEVS specification. System models using the SysML modeling language were transformed into DEVS executable models because the SysML model is not simulation-specific G. Kapos and Anagnostopoulos.. In Y. Lei and Zhu., the authors developed a simulation model using Simulation Model Definition Language (SMDL). In this work, the simulation model using DEVS was transformed into the standard SMP2.

From the perspective of AADL models verification, using the model checking techniques for this purpose can be challenging Hamdane *et al.* (2013). AADL models are therefore often transformed into a different verification formalism. For instance, M. Chkouri and Sifaksi. (2008) described the translation of AADL to BIP, which allows the simulation of AADL models. The transformation of AADL to timed automata is proposed in Hamdane *et al.* (2013).

CHAPTER 3

TIME-TRIGGERED ETHERNET METAMODEL

In this chapter we present a model-based approach to model the IMA architecture combined with the new SAE standard TTEthernet as communication infrastructure provide a strong platform for the deployment of distributed avionic applications. For that we begin with introducing the metamodel of TTEthernet and then in next chapter we explain implementation of this metamodel.

The verification approach for AADL-TTEthernet metamodel that has been used in this thesis contains three major steps, reported as follow:

a. Presentation and discussion with our industrial partners in order to assure about correctness and ability to model different safety critical application such as avionic system deployed on IMA architecture interconnected with TTEthernet.

b. Presentation and discussion with SAE AADL committee. we are member of this committee that organize periodic meeting every three months. We improve our metamodel based on the feedback reached from their experts.

c. Conference and journal paper published about AADL-TTEthernet metamodel that validate the correctness and efficiency of AADL-TTEthernet metamodel.

3.1 TTEthernet Model

In this chapter, we introduce our proposal for a metamodel of TTEthernet which represent our first original contribution for this thesis. This metamodel is based on SAE Standard Aerospace (2011d), which provides time-triggered services. These services are attached to the Ethernet. To better understand TTEthernet services, we present a model for TTEthernet. TTEthernet is then mapped to the metamodel of TTEthernet that supports SAE AS6802 Standard Aerospace (2011d).

Based on AG. (2009), a TTEthernet model can be structured in different hierarchical levels, as shown in Table 3.1. These levels are *network* level (Figure 3.1), *multi cluster* level (Figure 3.2) and *cluster* level (Figure 3.3).

Table 3.1 TTEthernet levels

Level	**Synchronization Domain**	**Synchronization Priority**
Network	Y	(Y,Z)
Multi Cluster	1	X
Cluster	1	1

Figure 3.1 presents an example of a TTEthernet *network* composed of four *multi clusters*. This TTEthernet *network* presents Y for the different *synchronization domains* per *multi cluster*, and (Y,Z) for the different *synchronization priorities* per *cluster*, where Y represents the priority of a *multi cluster*, and Z the priority of the *clusters* inside the *multi cluster*.

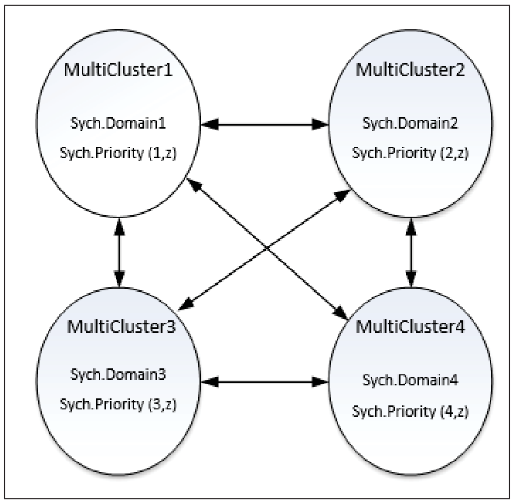


Figure 3.1 TTEthernet network

A *multi cluster* refers to an independent TTEthernet system containing many *clusters*, as shown in Figure 3.2. Each *cluster* belongs to the same *synchronization domain* as the *multi cluster*, but it has different *synchronization priorities*, which are represented by X in Table 3.1.

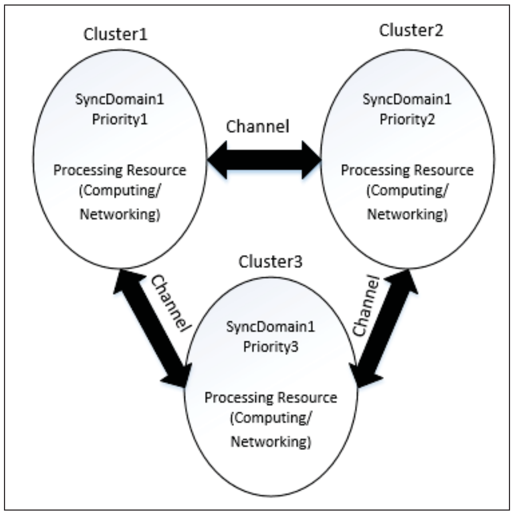

Cluster1
SyncDomain1
Priority1
Processing Resource (Computing/ Networking)
Cluster2
SyncDomain1
Priority2
Processing Resource (Computing/ Networking)
Cluster3
SyncDomain1
Priority3
Processing Resource (Computing/ Networking)
Channel
Channel
Channel

Figure 3.2 TTEthernet Multi Cluster

A *cluster* refers to a TTEthernet component that contains multiple *Processing Resources*. These *Processing Resources* can be either *Computing Resources* or *Networking Resources*, as shown in Figure 3.3. In a distributed system, the *Computing Resources* (e.g. End-Systems [ESs]) are the active components of the system. The *Networking Resources* (e.g. switches) are responsible for communication throughout a distributed system.

Processing Resources can play the role of *Synchronization Master (SM)*, *Compression Master (CM)* or *Synchronization Client (SC)*. The *SM* transmits its local time encapsulated in a *Protocol Control Frame (PCF)* to the *CM*. The *CM* collects the *PCF* from the *SM* and then dispatches a new PCF. The *SMs* and *SCs* use the dispatch point of the new PCF to re-synchronize their local clocks. In fact, the *SC* consumes the *PCF* for synchronization purposes.

3.2 A Metamodel for TTEthernet Domain

The TTEthernet metamodel captures the main concepts and characteristics of the SAE TTEthernet standard. It enables the building of a set of tools to perform the design and analysis of distributed architectures using TTEthernet as a communication infrastructure. A global overview of the TTEthernet metamodel is shown in Figure 3.4. We have broken this metamodel down into sub-groups in order to provide a clearer description.

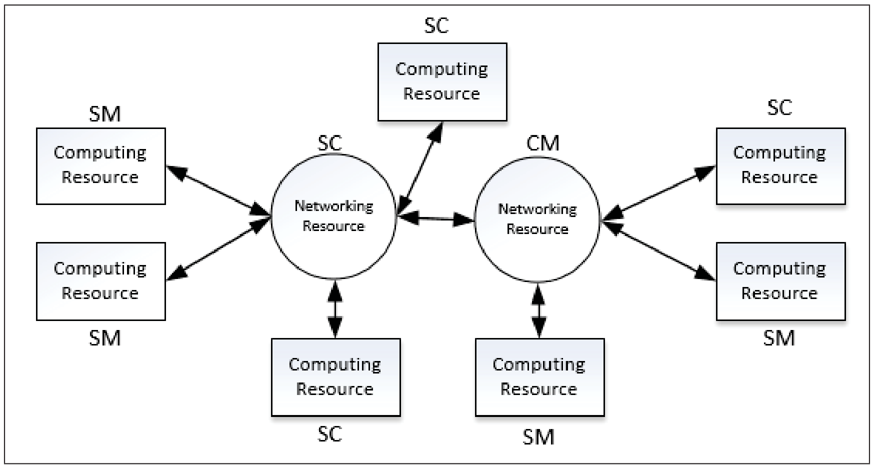


Figure 3.3 TTEthernet Cluster

Figure 3.5 presents an overview of the TTEthernet metamodel. The *TTEthernet MetaModel* class is composed of *Processing Resources*, where each *Processing Resource* could be a *Synchronization Master*, *Compression Master* or *Synchronization Client*. These *Processing Resources* are also divided into two classes: *Networking Resources* (e.g. *Switches*) and *Computing Resources* (e.g. *End-Systems*). At a high level of abstraction, the TTEthernet metamodel classes presented satisfy the requirements for modeling TTEthernet *cluster*. A *cluster* is composed of at least two *computing resources* that communicate together through a *networking resource*. In fact, the only missing part for modeling a *cluster* is the connection betweennetworking *Resources* and *computing Resources*. This is covered in Figure 3.6, which presents the concept of *virtual link*. A virtual link is a logical link that connects one source End-System to one or more destination End-Systems.

The *Synchronization Domain* class of the TTEthernet metamodel provides a specific domain of synchronization for a cluster. Every *synchronization domain* has its unique priority established by the *Synchronization Priority* class of the metamodel. A *cluster* supports only one *Synchronization Priority* and one *Synchronization Domain*. A *multi cluster*, which is composed of at least two *clusters*, presents one *Synchronization Domain* and multiple *Synchronization Priorities*. This is supported by an *EReference* between the *cluster class* and the *Synchronization Domain* of the metamodel. Finally, the TTEthernet *network* shown in Figure 3.1 is represented

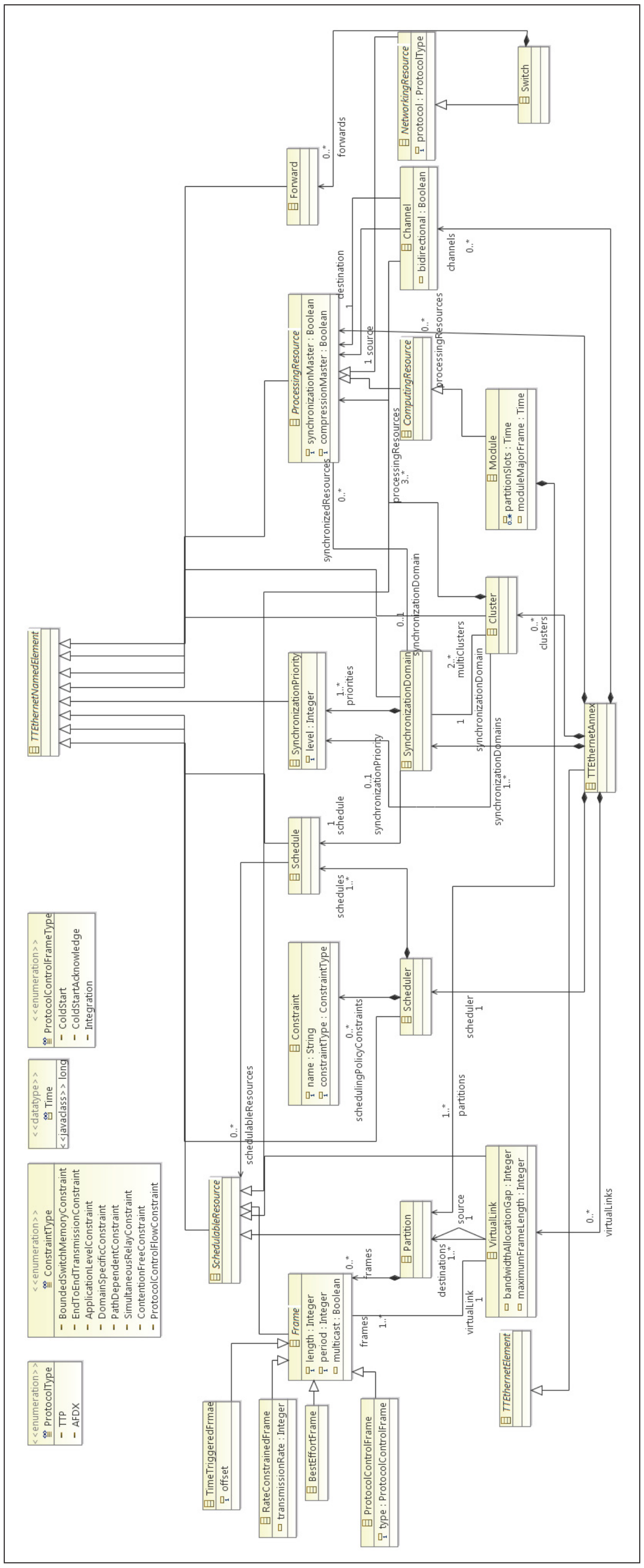


Figure 3.4 TTEthernet Metamodel

by the *TTEthernet MetaModel* class, which can have multiple *Synchronization Domains* and multiple *Synchronization Priorities*.

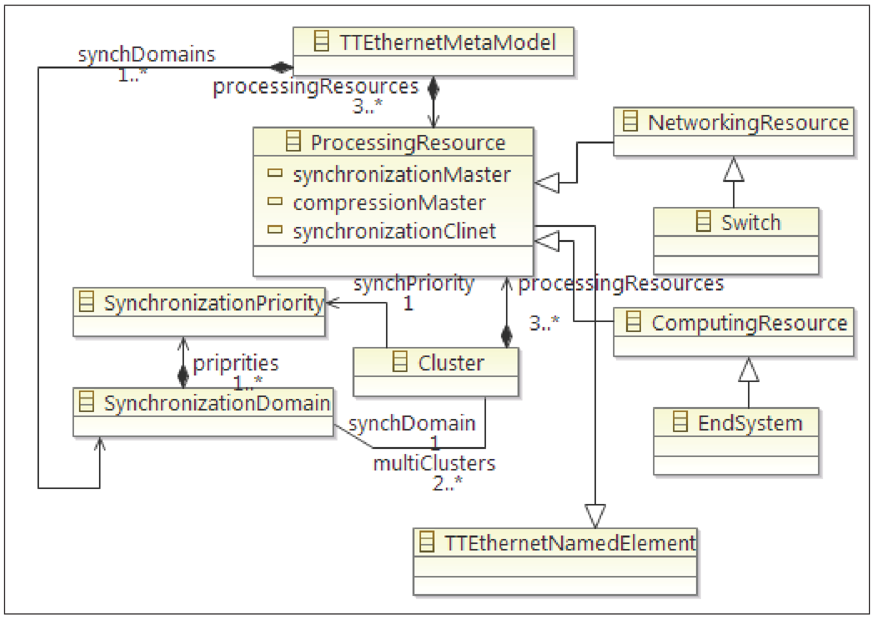


Figure 3.5 TTEthernet Metamodel Overview

3.2.1 Schedulable Resources

Having presented the main elements of the TTEthernet metamodel, here we take a closer look at the other essential elements of the metamodel. Figure 3.6 illustrates the *Schedulable Resources* of the TTEthernet network, which represents all of the elements related to the scheduler. These resources include the *End System*, *Frame*, *Channel* and *Virtual Link*. *End-Systems* are the nodes of a distributed system that performs the functionality of the system. To do this, every *End-System* hosts at least one system *Functionality*. This is represented by an*EReference* between the *End-System* class and *Functionality* class of the metamodel. In the case of an *End-System* that hosts multiple *Functionalities*, a strict isolation between them is required. The maximum number of *Functionalities* that can be dedicated to an *End-System*, is determined by the resource allocation function. *Frames* refer to data units that travel through the network. A *Virtual Link* is a logical connection that builds a communication tree between a source *End-*

System and multiple destination*End-Systems*. Each *Frame* belongs to a *Virtual Link*, while a *Virtual Link* can carry many *Frames*. This is indicated with an *EReference* between the *Frame* class and *Virtual Link* class of the metamodel. A *Functionality* must have only one *Virtual Link* as its source and one or many *Virtual Links* as its destination. This is supported by a bi-directional *EReference* between the *Functionality* class and *Virtual Link* class.

3.2.2 Channel

A *Channel* is a logical connection defined within the scope of a *cluster* or *multi cluster*. In fact, a *Channel* is provided to facilitate communication between *clusters* and *multi clusters*. *Frames* belong exclusively to one *Channel*, but *End-Systems* and their dedicated *Functionalities* can use multiple *Channels*.

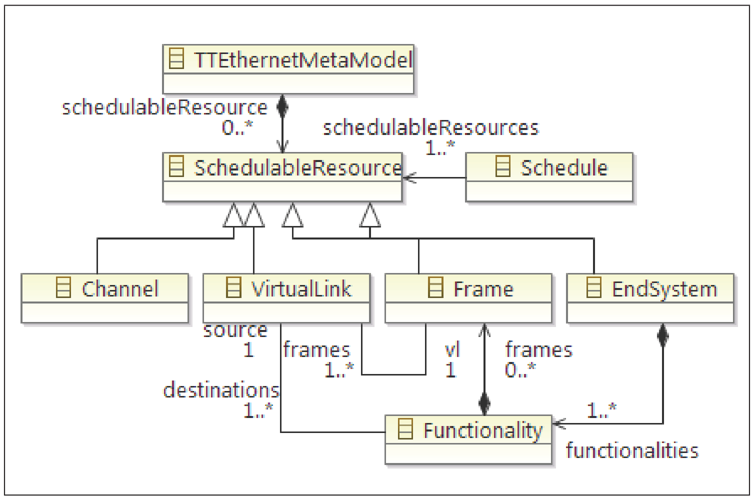


Figure 3.6 Schedulable Resources

3.2.3 Frames

As discussed in the background section, TTEthernet presents three types of *Frames*, each with different priorities, as shown in Figure 3.7. The attributes of each class of the metamodel in this figure, present the characteristics of a corresponding *Frame* type (e.g. *TTEthernet* class has *offset*). The *PCF* class, which is also a *Frame* type, supports the requirements of the synchronization protocol of TTEthernet.

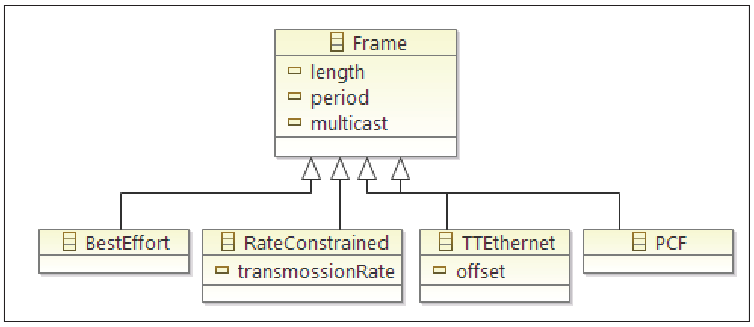


Figure 3.7 Frame Categories

3.2.4 Scheduler

The temporal communication behavior of the *Frames* is described in a *Schedule*. A *Scheduler* is a tool that produces a *Schedule*, which must respect specific constraints in order to support the TTEthernet communication paradigm. Figure 3.8 shows the *Schedule* and *Scheduler* classes of the metamodel, including the enumeration list of presented constraints. The constraints that have been reported in Steiner (2010) are the following: *Contention Free, Simultaneous Relay, Path Dependent, Domain Specific, Application Level, End To End Transmission, Bounded Switch Memory and Protocol Control Flow*. TTEthernet provides unique *Schedule* for the whole network even though the network is made up of several *multi clusters* and *Synchronization Domains*.

3.2.5 The Metamodel of IMA

Figure 3.9 shows the metamodel of IMA architecture. The presented definition of IMA conforms to this metamodel. An IMA architecture defines the relationship between the hardware (physical) and the software (logical) components. It is composed of a set of modules that communicate together through sets of switches and links.

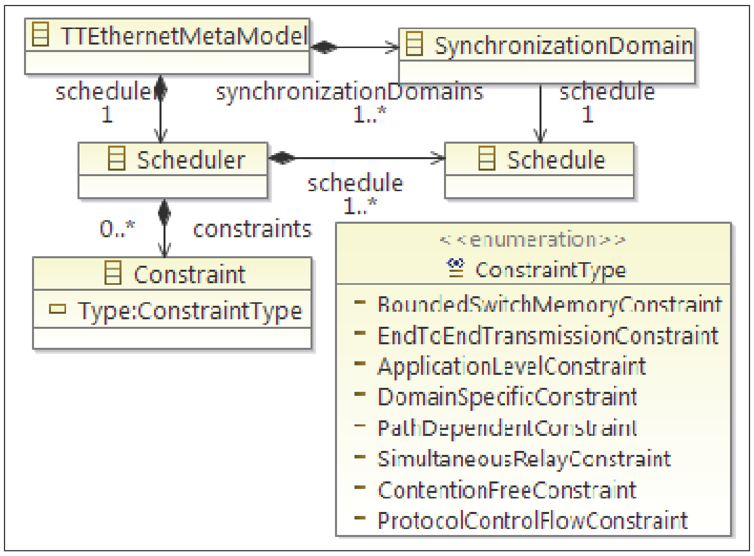


Figure 3.8 TTEthernet Scheduler

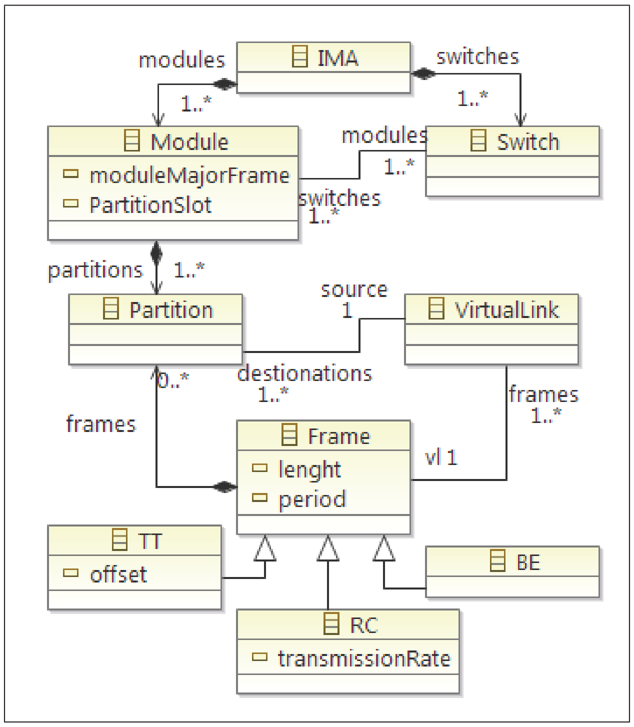


Figure 3.9 The metamodel of IMA

Table 3.2 shows how the IMA metamodel presented in Figure 3.9 is mapped onto the TTEthernet metamodel. In fact, the TTEthernet metamodel captures the main components of the IMA metamodel and then provides a model that represents IMA architecture using TTEthernet.

Table 3.2 IMA metamodel in accordance with TTEthernet metamodel

IMA metamodel:	**TTEthernet metamodel:**
IMA	Cluster
Module	End-System
Partition	Functionality
Frame	Frame
Switch	Switch
Virtual link	Virtual link

3.3 Conclusion

In this chapter, we have presented Integrated modular avionics architectures combined with the emerging SAE TTEthernet standard provides a strong infrastructure for the deployment of mixed-critical avionic applications having stringent safety, reliability and performance requirements. The integration of such systems is a very complex and challenging engineering task. Therefore, a model-based approach, which endows system engineers with a methodology and the supporting tools to cope with this complexity, is of a paramount importance.

In next chapter, we present the implementation process of TTEthernet metamodel using the SAE standard architecture language AADL which is achieved by the implementation of the OSATE2 extension mechanism.

CHAPTER 4

AN EXTENSION FOR AADL TO MODEL MIXED-CRITICAL AVIONIC SYSTEMS DEPLOYED ON IMA ARCHITECTURES WITH TTETHERNET

In this chapter we present an extension for the AADL modeling language to support modeling TTEthernet-based distributed systems. This extension consists essentially in a metamodel of the TTEthernet standard which has been presented in previous chapter and the implementation of its corresponding concrete syntax.

4.1 Metamodel Extending AADL capability to model TTEthernet

TTEthernet supports safety-critical applications due to its transparent integration of TT traffic and capacity to integrate different traffic types on the same physical platform. The TTEthernet standard specifies a synchronization strategy that establishes a global synchronized time in a distributed system. The standard focuses only on the network aspects of distributed systems, not their integration procedures. Thus, our metamodel for TTEthernet also covers the TTEthernet communication network in distributed systems.

In order to extend AADL with our metamodel, a TTEthernet model must be attached to an AADL component and the objects of our TTEthernet extension linked with AADL core objects. This is achieved by the implementation of the OSATE2 extension mechanism, which is needed to link the TTEthernetAnnex concept in our metamodel to the AnnexSubclause concept of the AADL core, as shown in Figure 4.1. This figure also shows how we use the EMF/Ecore inheritance mechanism to express the dependencies between the two metamodels. Consequently, a *TTEthernetAnnex* extends an *AnnexSubclause* and a *TTEthernetNamedElement* extends a *NamedElement*. In the metamodel, the TTEthernetAnnex concept, which links the metamodel to the AADL core metamodel, (shown in Figure 4.1) represents the overall model of a TTEhernet-networked IMA system that undergoes different analyses to verify its safety and performance properties. The global information about the network elements and the underlying implementation is described in the *TTEthernetAnnex* concept.

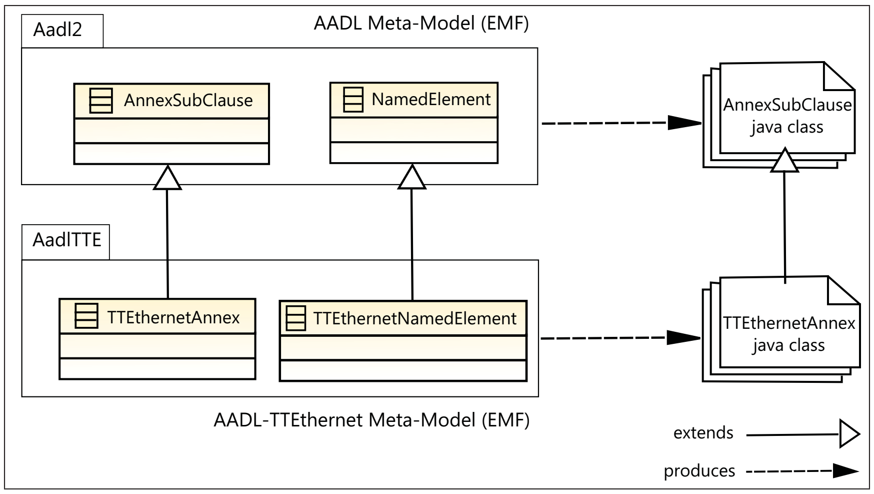


Figure 4.1 AADL-TTEthernet metamodel dependencies

4.2 Textual Syntax for the TTEthernet Extension for AADL

The definition of a textual syntax is provided by a grammar (i.e. a set of rules that defines the composition of a language). In order to translate the textual syntax into its corresponding model, a lexer, a parser and a component for semantic analysis (type checking, resolving of references, etc.) are required. The backward transformation, from model to text, is provided by an emitter. All of the components can be generated using the grammar $\Longleftrightarrow$ metamodel mapping definition Goldschmidt *et al.*. Figure 4.2 shows the selected framework to define the textual syntax of our extension. It employs the data provided by the mapping definition used to generate the parser, emitter and editor for the corresponding language of the metamodel. This editor can then use the generated parser and emitter to modify the text and the model. Thus, it is responsible for keeping the text and model in sync (e.g. by calling on the parser if there are any changes in the text). Based on this mapping definition, several features of the editor can be activated, including syntax highlighting, autocompletion and error reporting.

To build the text editor tool of our AADL-TTEthernet extension, we used the xText framework Efftinge (2006). It implements textual syntax according to an extended BNF. Figure 4.3 shows an excerpt of this xText grammar. In this xText framework, the AADL-TTEthernet metamodel concept is mapped to a Java implementation, where the TTEthernet object names are used

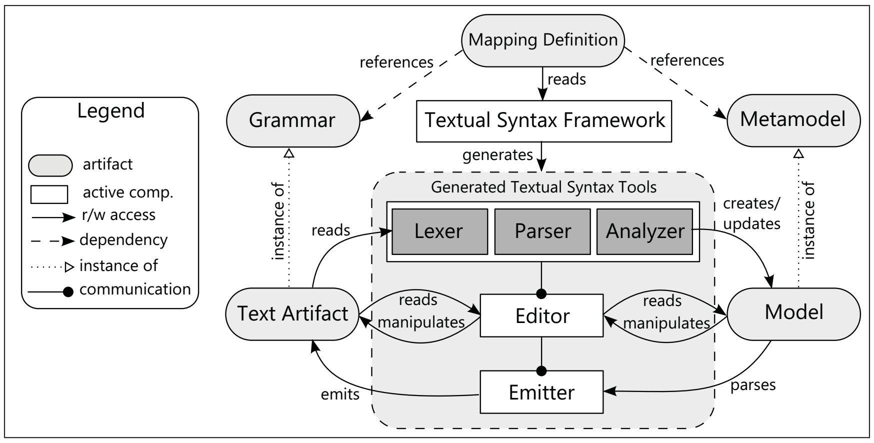


Figure 4.2 General structure of a textual syntax framework

as class names. All attributes are implemented as private fields as well as public get- and set- methods. The composition relationships are realized in the same way as attributes and contribute to the constructor of the class. All classes support the Visitor pattern Gamma *et al.* (1995) to traverse the abstract syntax along the composition relationships, Krahn *et al.* (2007).

The analyzer module scans the Abstract Syntax Tree (AST) and checks the semantics of the AADL-TTEthernet model. First, it proceeds to a resolution phase (e.g. naming resolver), which links TTEthernet objects to their corresponding AADL objects. In order to accomplish this, we used the visitors (e.g. java classes) provided by OSATE2 to retrieve AADL objects. For the sake of the implementation of our AADL-TTEthernet extension, we developed the visitors required to navigate through the AADL-TTEthernet AST. This phase adds information to the AST and makes it easier to use.

4.2.1 Integration of the AADL-TTEthernet Compiler to OSATE2

Sub-languages are included with AADL specifications as annex subclauses. The latter may be inserted into AADL component types and AADL component implementations of an AADL model. OSATE2 currently provides four extension points that can be used to integrate a sub-language into the tool environment. These extension points are designed to support parsing, unparsing, name resolution / semantic checking, and instantiation of annex models. From

```
grammar org.osate.ttethernet.xtext.Aadltte

import "http://ca.estmtl.aadl2/aadltte/1.0"
import "http://aadl.info/AADL/2.0" as aadl2
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Partition returns Partition:
        'Partition' name = ID
                        'frames' ':'frames += Frame*
        'end' ID ';';
Frame:
        RateConstrainedFrame | TimeTriggeredFrame | BestEffortFrame
   | ProtocolControlFrame
;
SynchronizationPriority returns SynchronizationPriority:
        'Synchronization Priority' name = ID
                'level' level = Integer ';'
        'end' ID ';';
```

Figure 4.3   xText grammar overview for AADL-TTEthernet

the AADL-TTEthernet EMF meta-model in the EMF framework, we generate the AADL-TTEthernet builder factory to build and manipulate TTEthernet objects used in the compiler. The compiler plug-in contains two modules: a parser/lexer and an analyzer. The integration of the AADL-TTEthernet plug-in is a two-step process. First, we link the AADL-TTEthernet plug-in to the OSATE2 annex plug-in using the Eclipse extension points mechanism. The annex plug-in defines extension points, which allows the plug-ins to be connected together, as shown in Figure 4.4. Second, we have to register our parser in the OSATE2 annex registry. Since the AADL-TTEthernet metamodel becomes a part of the AADL description, and the AADL-TTEthernet textual syntax tool is connected to the OSATE2 registry, the AADL-TTEthernet plug-in is directly integrated and driven by OSATE2.

4.3 An Example: A Model of a Subsystem of the Flight Management System

In this section, we illustrate the modeling of a distributed IMA system using TTEthernet as a communication network with our extension for AADL. To do this, we used a subsystem of the Flight Management System presented in M.Lauer (2012). This subsystem controls the display of static navigation information in the cockpit screens. The structure of this FMS subsystem, along with the modules and partitions they host, is shown in Figure 4.5. In the original version of the system presented in M.Lauer (2012), the system is interconnected using AFDX.

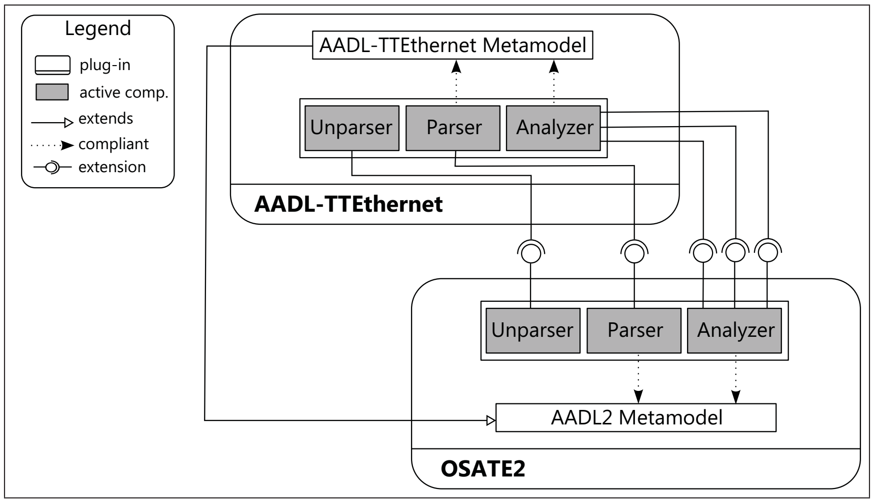


Figure 4.4 AADL-TTEthernet plug-in integrated to OSATE2

In our context, the modules are interconnected using TTEthernet instead. The AFDX data traffic in the original system corresponds to the RC traffic in the TTEthernet context. Table 4.1 shows a subset of the virtual links used in the FMS subsystem with their corresponding characteristics including the Bandwidth Allocation Gap (BAG), the sender modules of the VLs and the corresponding receiver modules.

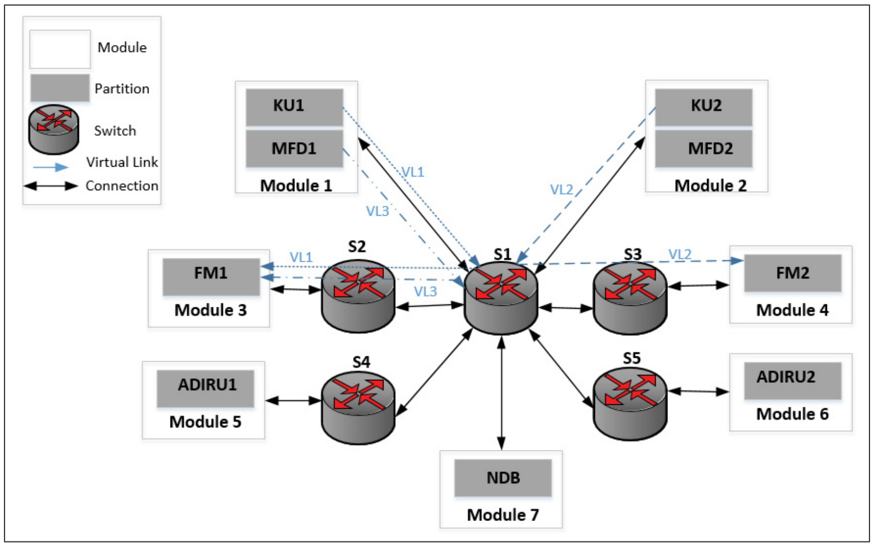
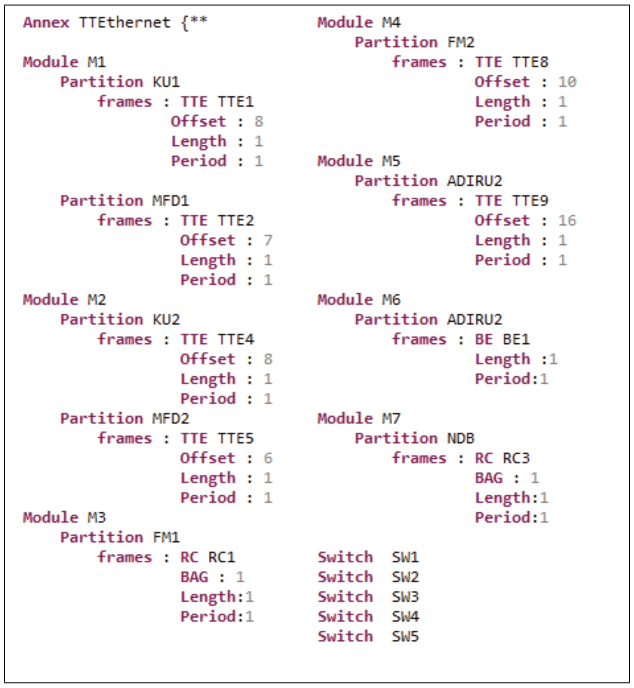


Figure 4.5 A subsystem of the Flight Management System

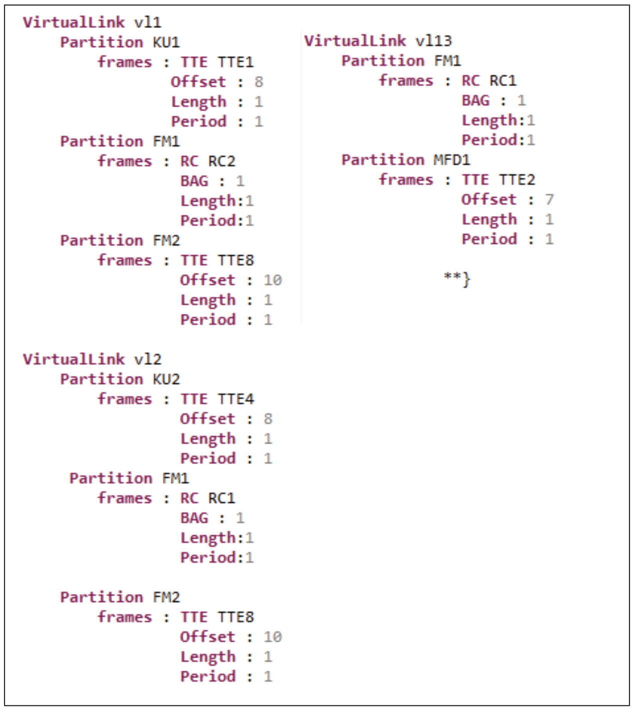
Table 4.1 Virtual Links details

Virtual Link	Source	Destination	BAG	Direction
VL_1	KU_1	FM_1, FM_2	32	$\{S_1, S_2\}$, $\{S_1, S_3\}$
VL_2	KU_2	FM_1, FM_2	32	$\{S_1, S_2\}$, $\{S_1, S_3\}$
VL_3	FM_1	MFD_1	8	$\{S_2, S_1\}$

This subsystem can thus be modeled using our TTEthernet extension for AADL. The extension is a sub-language for AADL, which can be included in the *system implementation* of the AADL model of this system. The concrete textual syntax of the AADL-TTEthernet extension provides several new reserved words, which correspond to the main concepts of the metamodel described previously (e.g. module, switch, partition, connection, virtual link, Time-triggered frame, Rate Constraint frame and Best Effort frame). An excerpt of the model of the FMS subsystem using our AADL-TTEthernet is shown in Figure 4.6.



```
Annex TTEthernet {**

Module M1
    Partition KU1
        frames : TTE TTE1
                Offset : 8
                Length : 1
                Period : 1

    Partition MFD1
        frames : TTE TTE2
                 Offset : 7
                 Length : 1
                 Period : 1
Module M2
    Partition KU2
        frames : TTE TTE4
                 Offset : 8
                 Length : 1
                 Period : 1
    Partition MFD2
        frames : TTE TTE5
                 Offset : 6
                 Length : 1
                 Period : 1
Module M3
    Partition FM1
        frames : RC RC1
                 BAG : 1
                 Length:1
                 Period:1

Module M4
    Partition FM2
        frames : TTE TTE8
                 Offset : 10
                 Length : 1
                 Period : 1

Module M5
    Partition ADIRU2
        frames : TTE TTE9
                 Offset : 16
                 Length : 1
                 Period : 1

Module M6
    Partition ADIRU2
        frames : BE BE1
                 Length :1
                 Period:1

Module M7
    Partition NDB
        frames : RC RC3
                 BAG : 1
                 Length:1
                 Period:1

Switch  SW1
Switch  SW2
Switch  SW3
Switch  SW4
Switch  SW5
```



```
VirtualLink vl1
    Partition KU1
        frames : TTE TTE1
                Offset : 8
                Length : 1
                Period : 1
    Partition FM1
        frames : RC RC2
                 BAG : 1
                 Length:1
                 Period:1
    Partition FM2
        frames : TTE TTE8
                 Offset : 10
                 Length : 1
                 Period : 1

VirtualLink vl2
    Partition KU2
        frames : TTE TTE4
                 Offset : 8
                 Length : 1
                 Period : 1
     Partition FM1
        frames : RC RC1
                 BAG : 1
                 Length:1
                 Period:1

    Partition FM2
        frames : TTE TTE8
                 Offset : 10
                 Length : 1
                 Period : 1
```

```
VirtualLink vl13
    Partition FM1
        frames : RC RC1
                 BAG : 1
                 Length:1
                 Period:1
    Partition MFD1
        frames : TTE TTE2
                 Offset : 7
                 Length : 1
                 Period : 1

               **}
```

Figure 4.6 Flight Management Subsystem Model using AADL TTEthernet Extension

4.4 Conclusion

In this chapter, we have presented an extension for the standard architecture and analysis modeling language AADL to enable modeling integrated multi-critical avionic applications deployed on TTEthernet-based IMA architectures. In particular, we have presented a metamodel which extends the core AADL metamodel with concepts and constraints relevant for this domain, we have defined the concrete textual syntax for this extension and we outline the implementation of this extension using the Open Source AADL Tool Environment (OSATE). Finally, we have illustrated our AADL extension using a case study based on the Flight Management System. In our ongoing research , we aim at formalizing this extension in the form of a new annex through the SAE standardization process.

In next chapter we aim at defining a formal semantics for our extension to allow transforming the AADL models built using our extension to models that are suitable for analysis techniques that can be used to verify relevant safety and performance properties. we leverage model transformations to enable undertaking the verification of the system models expressed using our AADL extension.

CHAPTER 5

VERIFICATION APPROACH TO THE DESIGN OF DISTRIBUTED IMA ARCHITECTURES USING TTETHERNET

In this chapter, we present the overall architecture of our approach for verifying distributed IMA systems using TTEthernet as a communication infrastructure. We chose DEVS formalism, which simulates the behavior of a system at discrete points in time. The DEVS metamodel is built on top of the Eclipse Modeling Framework (EMF) and DEVS formalism Sarjoughian and Markid (2012). This is helpful because our proposed metamodel for TTEthernet is also deployed on EMF, so it reduces the useless implementation process. The DEVS metamodel is divided into structural and behavioral parts, where both can be defined for atomic and coupled models.

The overall architecture of our approach for modeling and verifying distributed IMA systems using TTEthernet as a communication infrastructure is depicted in Figure 5.1.

The source system model is an AADL model using our AADL-TTEthernet extension. This model represents an avionic application deployed on a distributed IMA system interconnected using TTEthernet. This model is actually an instance of the AADL-TTEthernet metamodel, which is presented in Chapter 3. The second part, presented in Section 5.1, introduces and implements an approach for verifying the AADL-TTEthernet metamodel.This approach is based on an ATL model transformation Jouault *et al.* (2008) and followed by the DEVS simulation environment in order to verify and validate the properties of TTEthernet.

5.1 Verification Approach

In order to enable the simulation of the input system model, our approach is a classical M2M model transformation, which is now a well-established pattern in model-driven engineering France and Rumpe (2007); Mens and Van Gorp (2006). The target metamodel of this model transformation is a DEVS metamodel. We reuse a simplified version of this metamodel Sar-

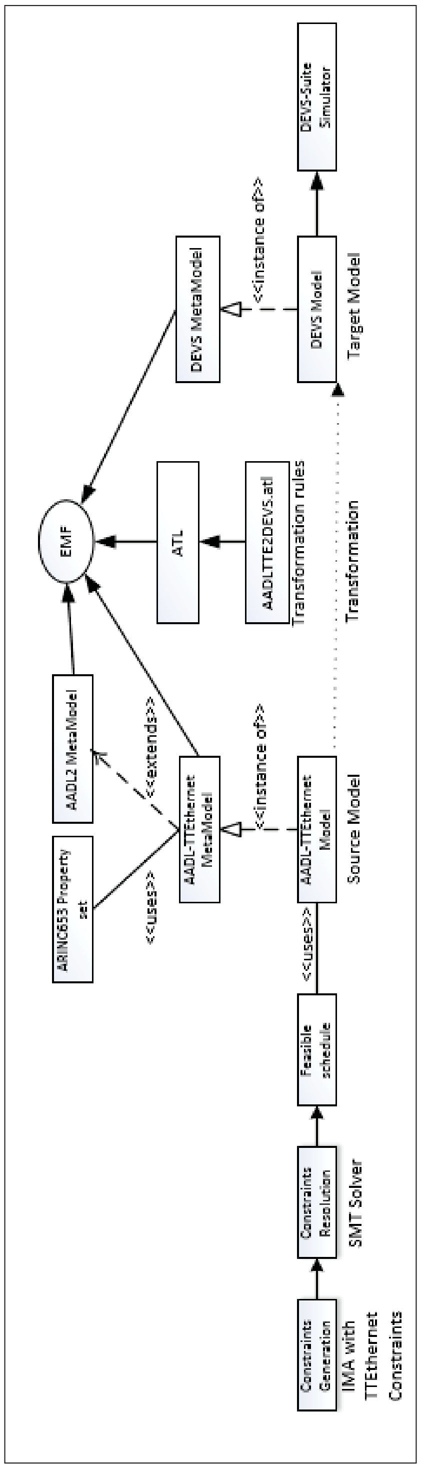


Figure 5.1 Overall Architecture

joughian and Markid (2012), which is shown in Figure 5.2 taken from Sarjoughian and Markid (2012). In this metamodel, *Atomic* and *Coupled* are two levels that DEVS formalism provides for the description of system behavior.

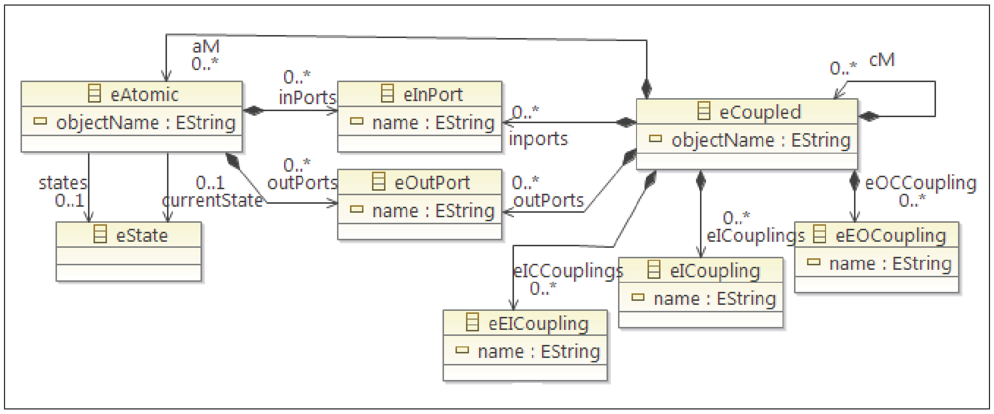


Figure 5.2 DEVS metamodel

At the lowest level, an *eAtomic* DEVS describes the autonomous behavior of a system as the Finite State automata. Furthermore, it describes the way the *eAtomic* reacts to external inputs in order to generate the outputs. At the higher level, an *eCoupled* DEVS describes the system as a network of coupled components. In the case of the latter, the *eCoupled* DEVS reports how components influence each other and how the output of a component can become an input of another one. The *eInput* and *eOutput* of the DEVS metamodel are assigned to model the input and output of the system. Other DEVS metamodel classes are used to model different possible situations such as when two *eCoupled* or two *eAtomic* or one *eAtomic* or one *eCoupled*) are combined to represent the entire system.

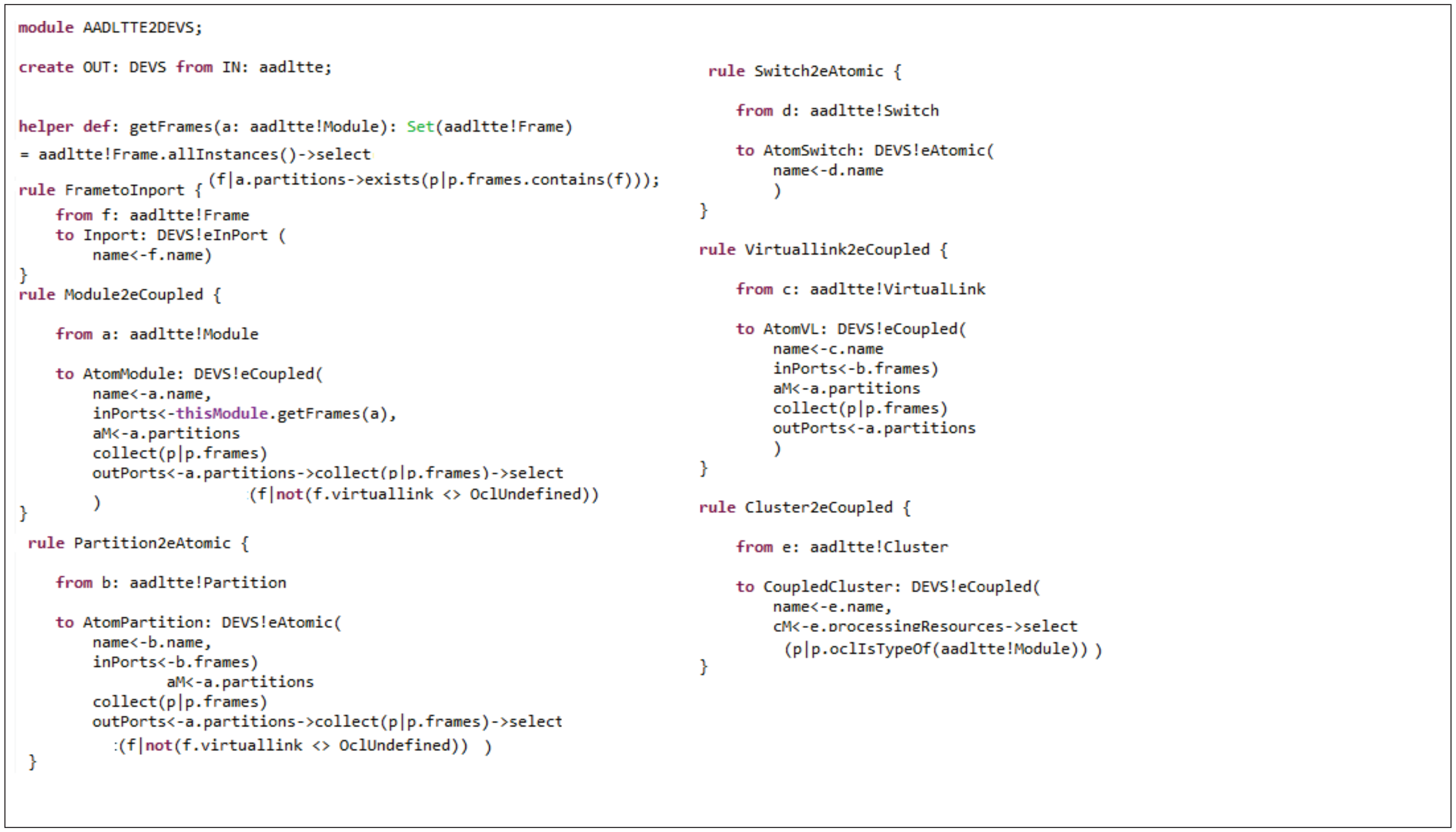
The transformation of an instance of the AADL-TTEthernet metamodel (i.e. the system model) into an instance of the DEVS metamodel (i.e. the simulation model) is achieved using a set of transformation rules specified in the ATL model transformation language F. Jouault and Kurtev (2008). These model transformation rules are based on the general mapping shown in Table 5.1. The Partition and Switch in the source metamodel are two entities that represent the behavior of the system. Therefore, they can be mapped into the *eAtomic* class of the DEVS metamodel. A

Module is mapped to the *eCoupled* class in order to connect the partitions it includes. A Cluster, which regroups modules and switches, is mapped to the *eCoupled*. A frame is the input data of the module, and the partition is mapped to the *eInport* of DEVS. A Virtual link is responsible for the coupling module, respecting partitions and switches, and is mapped to the*eCoupled* of DEVS metamodel. Thus, the ATL transformation rules given in Figure 5.3 specify how the module and partition concepts in the source metamodel are transformed into corresponding entities in the target metamodel. The target model, generated by the ATL transformation, is an intermediate model that can be used in the future to perform the model simulation realized by the DEVS simulation environment. The instance model that results from the transformation step needs to be refined in order to obtain a model that is suitable for simulation in the DEVS simulation environment. This refinement essentially consists of adding the behaviors of the source model to its implementation. To do this, we generate the JAVA code corresponding to the target model using Acceleo Eclipse, which is an implementation of the Model to Text Language (MTL) standard. The behavior of the source model is added to the JAVA code obtained with Acceleo.

Table 5.1 Mapping source model into target model

Source Model	**Target Model**
Cluster	eCoupled
Module	eCoupled
Partition	eAtomic
Frame	eInport
Switch	eAtomic
Virtual link	eCoupled

The main challenge in providing a model suitable for simulation with a hierarchical DEVS simulator is determining the sequence of the DEVS activation at run time. More specifically, this challenge involves the sequence of atomic or coupled in the entire simulatable model. We tackle this by means of a DEVS formalism *message-passing mechanism* Zeigler. (1984). The hierarchical DEVS simulator consists of the *DEVS simulator*, *DEVS coordinator* and *message-passing mechanism*, as shown in Figure 5.4. The *message-passing mechanism* shown in Figure



```
module AADLTTE2DEVS;

create OUT: DEVS from IN: aadltte;


helper def: getFrames(a: aadltte!Module): Set(aadltte!Frame)
= aadltte!Frame.allInstances()->select
                   (f|a.partitions->exists(p|p.frames.contains(f)));
rule FrametoInport {
    from f: aadltte!Frame
    to Inport: DEVS!eInPort (
        name<-f.name)
}
rule Module2eCoupled {

    from a: aadltte!Module

    to AtomModule: DEVS!eCoupled(
        name<-a.name,
        inPorts<-thisModule.getFrames(a),
        aM<-a.partitions
        collect(p|p.frames)
        outPorts<-a.partitions->collect(p|p.frames)->select
                        (f|not(f.virtuallink <> OclUndefined))
        )
}
 rule Partition2eAtomic {

    from b: aadltte!Partition

    to AtomPartition: DEVS!eAtomic(
        name<-b.name,
        inPorts<-b.frames)
                aM<-a.partitions
        collect(p|p.frames)
        outPorts<-a.partitions->collect(p|p.frames)->select
          :(f|not(f.virtuallink <> OclUndefined))  )
 }

 rule Switch2eAtomic {

    from d: aadltte!Switch

    to AtomSwitch: DEVS!eAtomic(
        name<-d.name
        )
}

rule Virtuallink2eCoupled {

    from c: aadltte!VirtualLink

    to AtomVL: DEVS!eCoupled(
        name<-c.name
        inPorts<-b.frames)
        aM<-a.partitions
        collect(p|p.frames)
        outPorts<-a.partitions
        )
}

rule Cluster2eCoupled {

    from e: aadltte!Cluster

    to CoupledCluster: DEVS!eCoupled(
        name<-e.name,
        cM<-e.processingResources->select
         (p|p.oclIsTypeOf(aadltte!Module)) )
}
```

Figure 5.3 ATL transformation rules

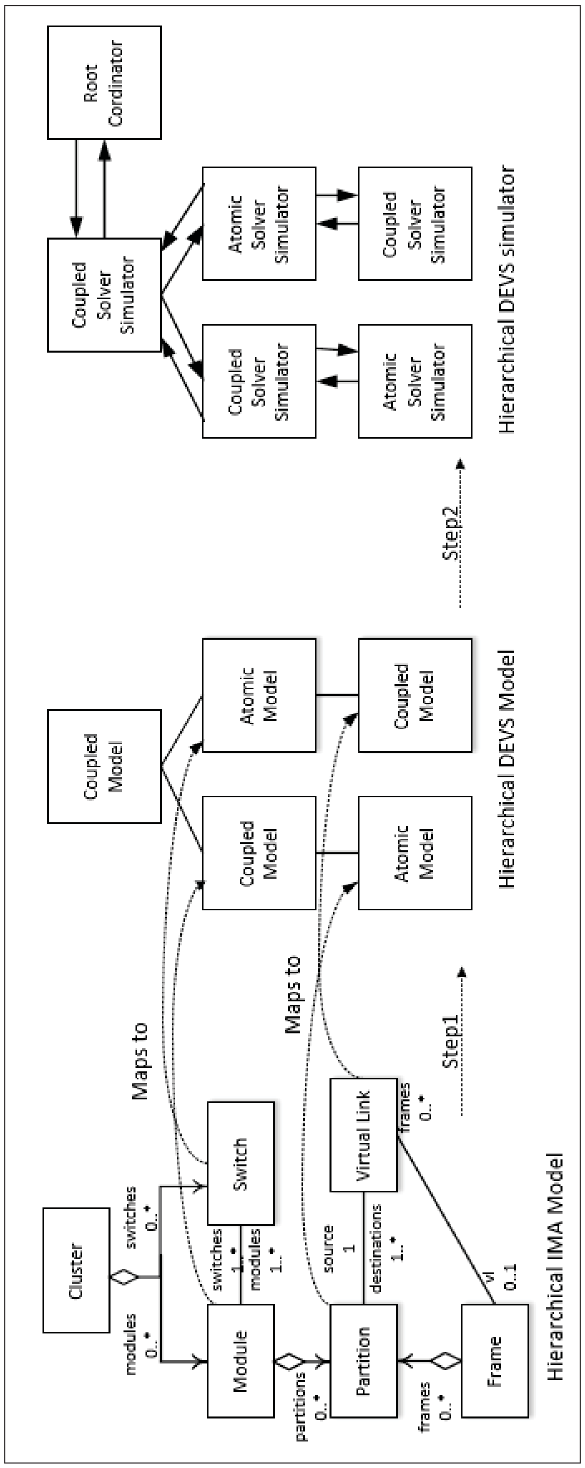


Figure 5.4 Mapping a hierarchical model onto a hierarchical simulator

5.4 with two directional arrows includes four categories of messages: an initialization method; an internal state transition message; an output message; and an input message to coordinator. This helps in controlling and monitoring the sequence of actions taken during the simulation. As illustrated in Figure 5.4, the mapping of a hierarchical IMA model to a hierarchical DEVS simulator is accomplished in Steps 1 and 2. The hierarchical IMA model in this figure represents an IMA architecture interconnected with TTEthernet. This model is mapped to the hierarchical DEVS model in Step 1. The hierarchical DEVS model is a DEVS model generated in accordance with the mapping rules. In Step 2, the hierarchical DEVS model is mapped to the hierarchical DEVS simulator, which is the simulatable model.

5.2 Simulation of the Navigation & Guidance System

In this section, we present a case study to illustrate our proposed approach. We present the system, its modeling using the proposed AADL extension, the transformation process of the model into a DEVS simulation model, and the verification of its contention free properties with the simulation model.

5.2.1 System Description

In this case study, we consider a simplified navigation and guidance system Lauer (2010b). As shown in Figure 5.5, the system is composed of four modules and two switches. The *Autopilot* (AP) module elaborates a flight command to reach an altitude defined by the next way-point of the flight plan. The *Multifunction Control Display Unit* (*MCDU*) presents an interface between the system and the crew. *Flight Management* (*FM*) periodically sends the next way-point*(pos)* to the *AP*. *Flight Warning* (*FW*) reports the equipment status *(sens-stat)* to the *MCDU*. Finally, the module *Anemometer* (*Anemo*) computes and broadcasts the speed (M) and the altitude (Z) to the *AP*. Z and M are two critical data that are encapsulated in TTEthernet frames. They are transmitted in two distinct frames, which are transmitted through VL_1 from the *Anemo* to the *AP* via SW_1 and SW_2.

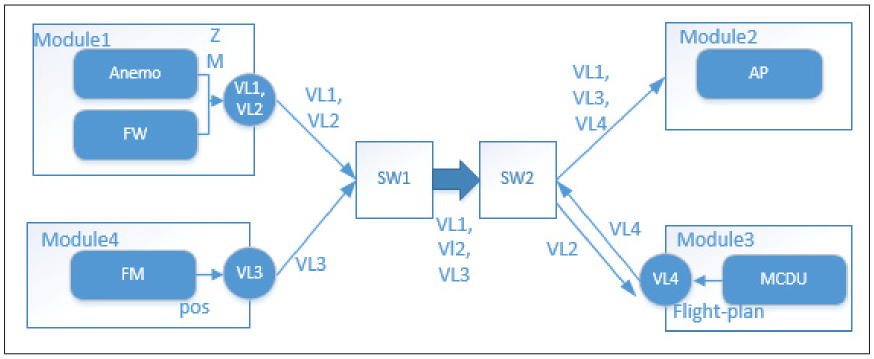


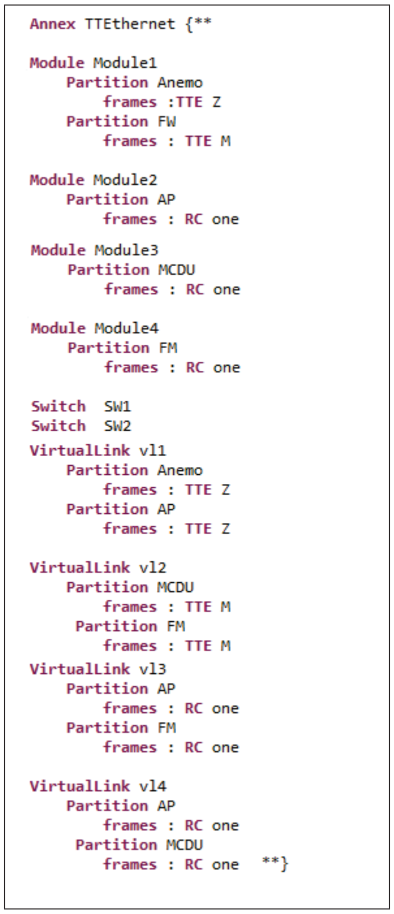
Figure 5.5 The Navigation & Guidance system

5.2.2 Model Transformation

The model of the navigation and guidance system using our AADL extension is given in Figure 5.6. This is an instance of the AADL-TTEthernet metamodel discussed previously. It is specified using the concrete textual syntax that we implemented in our proposed AADL annex for TTEthernet. Figures 5.7 and 5.8 show the internal representation of this model in the Eclipse EMF modeling framework and the corresponding target model (i.e. an instance of the DEVS metamodel) that is generated using model transformation Step 1 shown in Figure 5.4. The final model using the DEVS simulation environment is shown in Figure 5.9. This is the result of Step 2 shown in Figure 5.4, which essentially consists of adding the behavior of the source model to the JAVA code produced with Acceleo Eclipse.

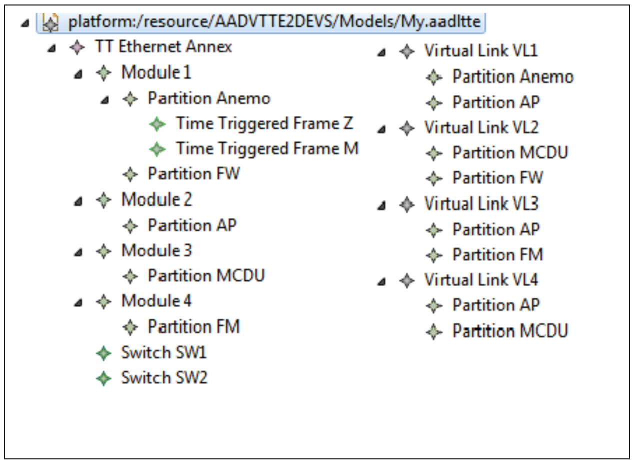
5.2.3 Property verification: Contention-Freedom

In this section, we illustrate the verification step in our proposed approach of the scheduling properties of TTEthernet. The schedule for a TTEthernet-based system needs to meet a specific set of constraints and properties defined in Steiner (2010). Here, we consider the fundamental constraint of the TTEthernet network known as *contention-freedom*. This constraint ensures the mutual exclusion of the frames transmitted in the same data flow link. This means that, within a given data flow link, only one frame can be transmitted at a time. Therefore, when a pair of frames are to be transmitted within a given link, the dispatch of the second frame will



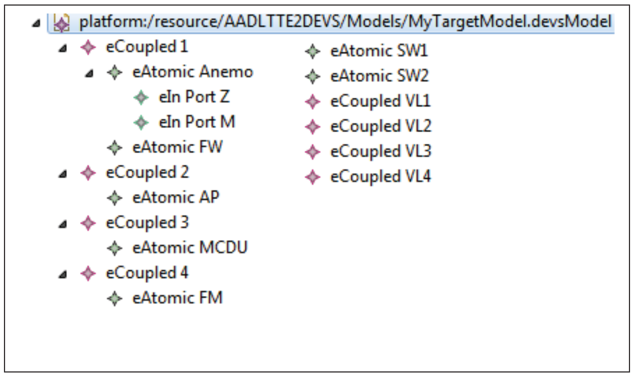
```
Annex TTEthernet {**

Module Module1
    Partition Anemo
        frames :TTE Z
    Partition FW
        frames : TTE M

Module Module2
    Partition AP
        frames : RC one

Module Module3
    Partition MCDU
        frames : RC one

Module Module4
    Partition FM
        frames : RC one

Switch  SW1
Switch  SW2
VirtualLink vl1
    Partition Anemo
        frames : TTE Z
    Partition AP
        frames : TTE Z

VirtualLink vl2
    Partition MCDU
        frames : TTE M
     Partition FM
        frames : TTE M
VirtualLink vl3
    Partition AP
        frames : RC one
    Partition FM
        frames : RC one

VirtualLink vl4
    Partition AP
        frames : RC one
     Partition MCDU
        frames : RC one   **}
```

Figure 5.6   Textual Syntax



```
platform:/resource/AADVTTE2DEVS/Models/My.aadltte
  TT Ethernet Annex
    Module 1
      Partition Anemo
        Time Triggered Frame Z
        Time Triggered Frame M
      Partition FW
    Module 2
      Partition AP
    Module 3
      Partition MCDU
    Module 4
      Partition FM
    Switch SW1
    Switch SW2
    Virtual Link VL1
      Partition Anemo
      Partition AP
    Virtual Link VL2
      Partition MCDU
      Partition FW
    Virtual Link VL3
      Partition AP
      Partition FM
    Virtual Link VL4
      Partition AP
      Partition MCDU
```

Figure 5.7   TTEthernet Model



- platform:/resource/AADLTTE2DEVS/Models/MyTargetModel.devsModel
  - eCoupled 1
    - eAtomic Anemo
      - eIn Port Z
      - eIn Port M
    - eAtomic FW
  - eCoupled 2
    - eAtomic AP
  - eCoupled 3
    - eAtomic MCDU
  - eCoupled 4
    - eAtomic FM
  - eAtomic SW1
  - eAtomic SW2
  - eCoupled VL1
  - eCoupled VL2
  - eCoupled VL3
  - eCoupled VL4

Figure 5.8 The corresponding DEVS model

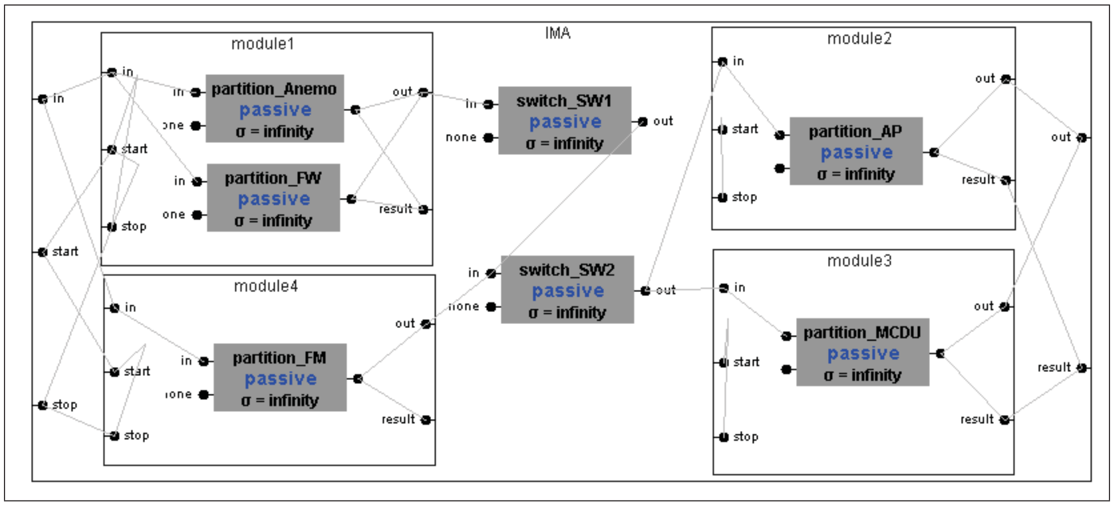


Figure 5.9 Simulation graph for the navigation & guidance system

either come after the transmission of the first, or the first will come after the transmission of the second. In order to verify the *Contention-freedom* property, we have to run two scenarios. In the first scenario, the schedule fulfills the contention-free constraint, and, in the second, the schedule violates this constraint. The generated simulation models *Job1* and *Job2* represent the TTEthernet frames Z and M, respectively, in the input model of the simplified navigation and guidance system. With the first scenario, *Job1* is dispatched at instant 10 and is received by Module 2 at instant 40. *Job2* is dispatched at instant 40 and is is received at instant 70 by Module 2. Therefore, the *contention-freedom* is verified with all jobs in the first scenario. However, for the schedule used in the second scenario, the dispatch time of *Job2* at instant 30 takes place before the reception time of *Job1* by Module 2 at instant 40, which violates the *contention-freedom* constraint (i.e. the transmission of frames Z and M would overlap in the same data flow link).

5.3 Conclusion

In this chapter we have presented a model transformation to enable the simulation of system models specified using the proposed AADL extension. The generated models with this transformation can be simulated in a DEVS simulation environment to check the model with respect to the required TTEhernet constraints. We have applied our approach to generate a simulation model starting with an AADL model of a simplified version of a navigation and guidance system and illustrated the verification of the compliance of the system schedule with the contention-free constraint.

In next chapter we apply over all our approach to a real case study provided by our industrial partner. Toward that, firstly we present the AADL model of this case study using our previously presented AADL extension. Then we provide verification process that we have presented in this chapter for the contention-free constraint of TTEthernet property.

CHAPTER 6

CASE STUDY

In this chapter, we use a case study to illustrate our proposed approach. We present the system and its modeling using the AADL extension that we proposed in this thesis. We explain the transformation process of the AADL model, corresponding to the case study, into a DEVS simulation model. Finally we illustrate the verification of the contention free property of TTEthernet using the simulation model. It is important to note that this case study was developed by Bombardier Aerospace and contains confidential information. Due to that, we present an abstract version of this example.

We aim at developing the model of this case study by using AADL-TTEthernet metamodel which is main original contribution of this thesis. In fact this case study is used to demonstrate how the proposed metamodel is capable to support IMA architecture interconnected with TTEthernet. In next step, the developed model is simulated using our simulation approach, which is our another original contribution.

6.1 System Description

In this case study, we consider an IMA architecture composed of different sub-systems. As shown in Figure 6.1, the system is composed of eight modules (ESs) and four switches (SWs). The ESs execute a different number of partitions based on their resources.

ES1 is composed of *Nose Landing Gear (NLG)*, *Gauging Channel A (GCA*) and *Fuel Mass (FM)*. Fuel Gauging provides details about the fuel, such as temperature, density and permittivity. Based on this data, the system computes the volume and mass of fuel available in each fuel tank.

ES2 is composed of *Main Landing Gear (MLG)*, *Gauging Channel B (GCB)* and *Fuel Transfer Center (FTC)*. *Nose Landing Gear (NLG)* provides commands for the *NLG* extension/retraction of actuators. *Right and Left Main Landing Gear (MLG)* provides commands for the *LMLG* and *RMLG* extension/retraction of actuators.

ES3 is composed of *Flight Deck Emulator (FDE)*, *LG & Fuel Aircraft Emulation (LGFAE)* and *SP Aircraft Emulator (SPAE)*. *LGFAE* emulates the behavior of actuators and provides status reports to the *NLG* and *MLG*.

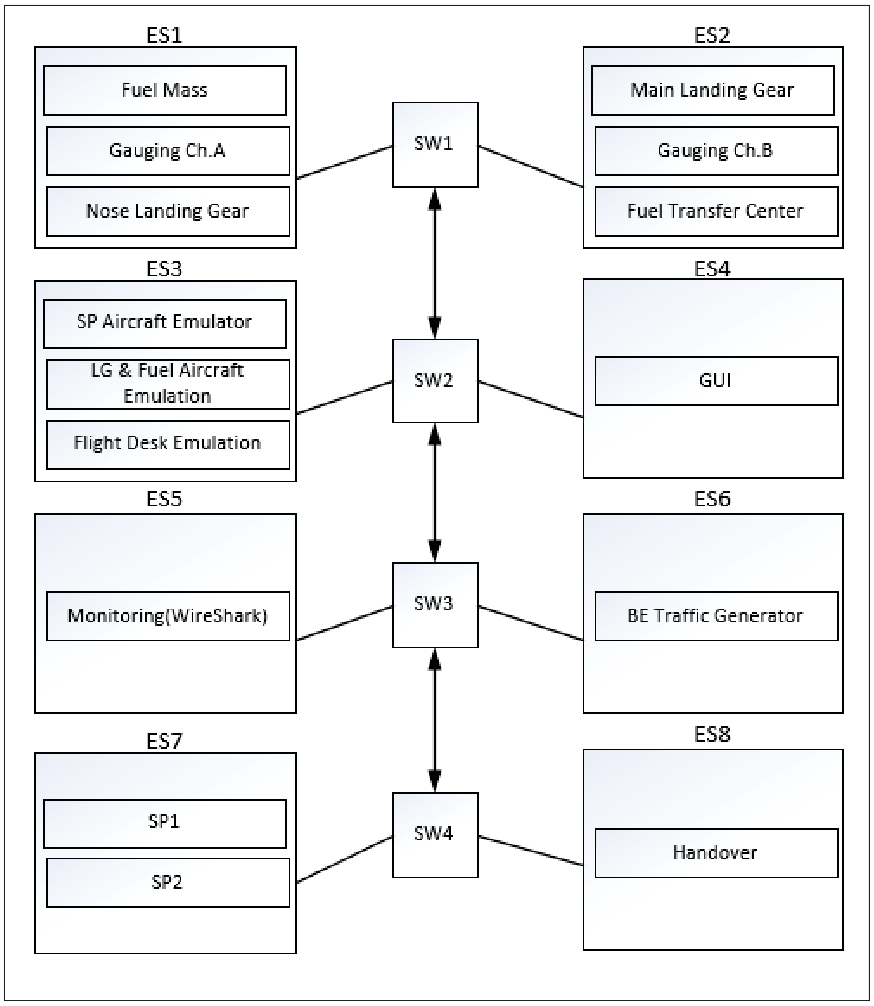


Figure 6.1   Case study

The communication between ESs is actuated through a large number of *Virtual Links (VLs)*. Table 6.1 demonstrates the properties of the VLs, which correspond to source and destination partitions, BAG and the maximum size of frames. Note that we only present some of these *VL*s here due to space constraints.

Table 6.1 Virtual Links detail

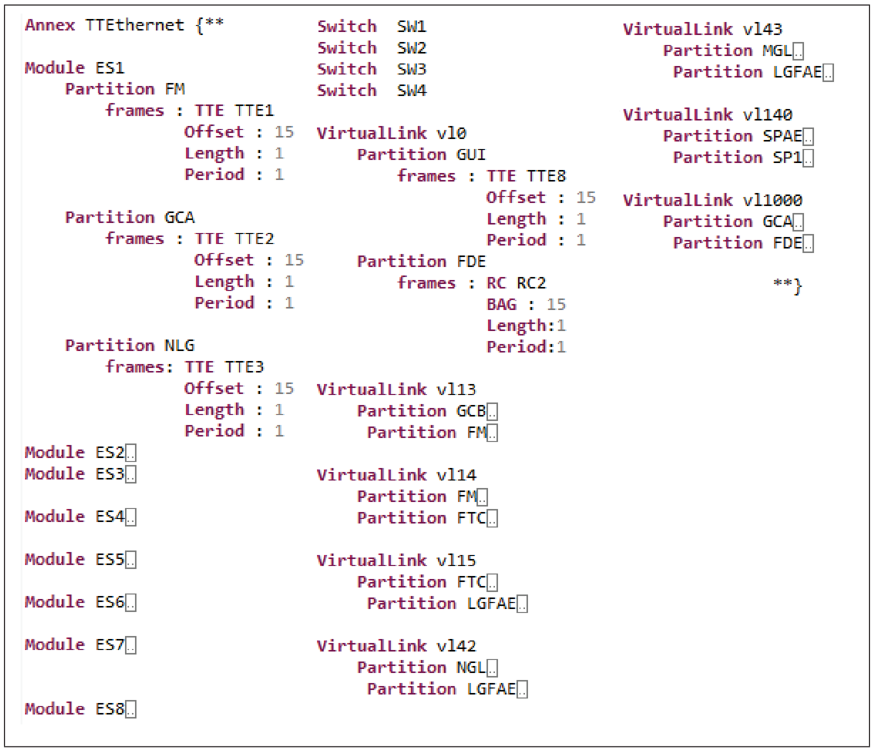
VLID	**Source**	**Destination**	**BAG**	**Max Frame Size**
0	GUI	FDE	15	114
13	GCB	FM	15	562
14	FM	FTC	15	562
15	FTC	LGFAE	15	114
42	NGL	LGFAE	15	114
43	MLG	LGFAE	15	114
140	SPAE	SP1	1	1518
1000	GCA	FDE	1	1518

The model of the case study using our AADL extension is given in Figure 6.2. This is an instance of the AADL-TTEthernet metamodel discussed above. It is specified using the concrete textual syntax that we implemented for our AADL Extension.

Figure 6.3 shows the internal representation of this model in the Eclipse EMF modeling framework, and Figure 6.4 shows the corresponding target model (i.e. an instance of the DEVS metamodel). The final model using the DEVS simulation environment is shown in Figure 6.5, which essentially consists of adding the behavior of the source model to the JAVA code produced with Acceleo Eclipse.

6.2 Property verification: Contention-Freedom

In this section, we explain the verification step of our proposed approach for the scheduling properties of TTEthernet. The schedule for a TTEthernet-based system needs to meet a specific set of constraints and properties defined in Steiner (2010). We consider here the fundamental constraint of TTEthernet network known as *contention-freedom*. This constraint ensures the mutual exclusion of frames transmitted in the same data flow link. This means that within a given data flow link, only one frame can be transmitted at a time. Therefore, if a pair of frames



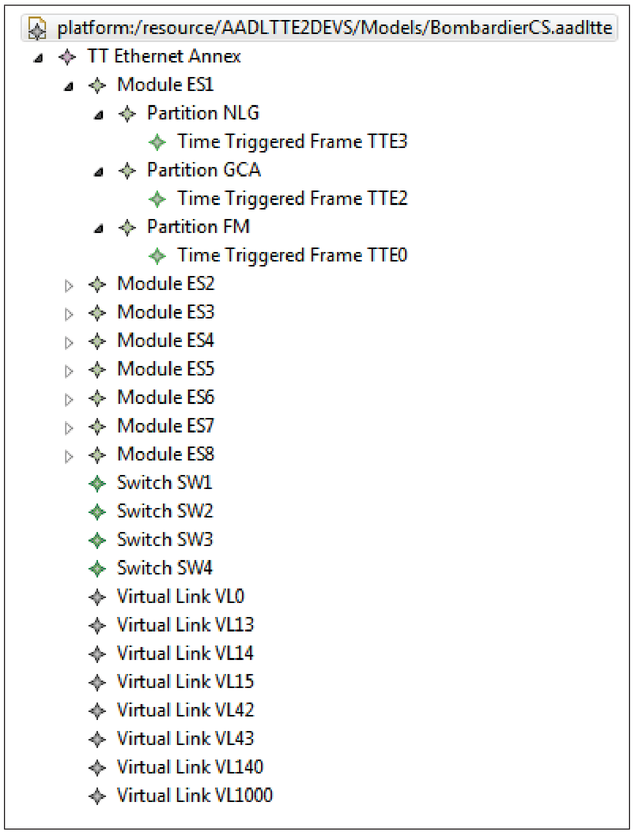
```
Annex TTEthernet {**

Module ES1
    Partition FM
        frames : TTE TTE1
                Offset : 15
                Length : 1
                Period : 1

    Partition GCA
        frames : TTE TTE2
                 Offset : 15
                 Length : 1
                 Period : 1

    Partition NLG
        frames: TTE TTE3
                Offset : 15
                Length : 1
                Period : 1
Module ES2⊡
Module ES3⊡

Module ES4⊡

Module ES5⊡

Module ES6⊡

Module ES7⊡


Module ES8⊡
```

```
Switch  SW1
Switch  SW2
Switch  SW3
Switch  SW4

VirtualLink vl0
    Partition GUI
        frames : TTE TTE8
                 Offset : 15
                 Length : 1
                 Period : 1
    Partition FDE
        frames : RC RC2
                 BAG : 15
                 Length:1
                 Period:1

VirtualLink vl13
    Partition GCB⊡
     Partition FM⊡

VirtualLink vl14
    Partition FM⊡
    Partition FTC⊡

VirtualLink vl15
    Partition FTC⊡
     Partition LGFAE⊡

VirtualLink vl42
    Partition NGL⊡
     Partition LGFAE⊡
```

```
VirtualLink vl43
    Partition MGL⊡
     Partition LGFAE⊡

VirtualLink vl140
    Partition SPAE⊡
     Partition SP1⊡

VirtualLink vl1000
    Partition GCA⊡
     Partition FDE⊡

                **}
```

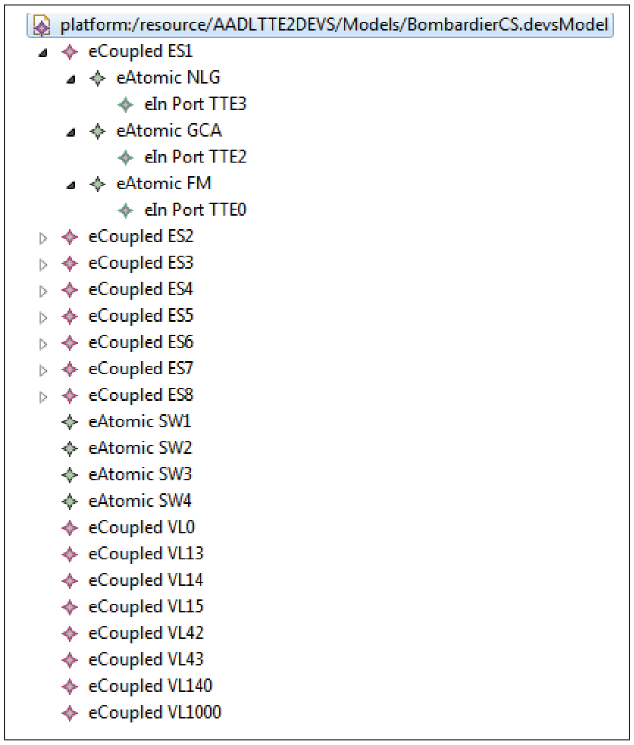
Figure 6.2 Textual Syntax

are transmitted within a given link, either the dispatch of the second frame will come after the end of the first, or the dispatch of the first will come after the second. In order to verify the *Contention-freedom* property, we have to run two scenarios. In the first scenario, the schedule fulfills the contention-free constraint, and, in the second, the schedule violates this constraint. In the generated simulation model, *Job1* and *Job2* are dispatched from ES1 and ES2. With the first scenario, *Job1* is dispatched at instant 10 and is received by ES2 at instant 40. *Job2* is dispatched at instant 40 and is received at instant 70 by ES2. Thus the *contention-freedom* is verified with both jobs in the first scenario. However, for the schedule used in the second scenario, the dispatch time of *Job2* at instant 30 takes place before the reception time of *Job1* by ES2 at instant 40 violating the *contention-freedom* constraint.



```
platform:/resource/AADLTTE2DEVS/Models/BombardierCS.aadltte
  TT Ethernet Annex
    Module ES1
      Partition NLG
        Time Triggered Frame TTE3
      Partition GCA
        Time Triggered Frame TTE2
      Partition FM
        Time Triggered Frame TTE0
    Module ES2
    Module ES3
    Module ES4
    Module ES5
    Module ES6
    Module ES7
    Module ES8
    Switch SW1
    Switch SW2
    Switch SW3
    Switch SW4
    Virtual Link VL0
    Virtual Link VL13
    Virtual Link VL14
    Virtual Link VL15
    Virtual Link VL42
    Virtual Link VL43
    Virtual Link VL140
    Virtual Link VL1000
```

Figure 6.3 The AADL TTEthernet model



```
platform:/resource/AADLTTE2DEVS/Models/BombardierCS.devsModel
  eCoupled ES1
    eAtomic NLG
      eIn Port TTE3
    eAtomic GCA
      eIn Port TTE2
    eAtomic FM
      eIn Port TTE0
  eCoupled ES2
  eCoupled ES3
  eCoupled ES4
  eCoupled ES5
  eCoupled ES6
  eCoupled ES7
  eCoupled ES8
  eAtomic SW1
  eAtomic SW2
  eAtomic SW3
  eAtomic SW4
  eCoupled VL0
  eCoupled VL13
  eCoupled VL14
  eCoupled VL15
  eCoupled VL42
  eCoupled VL43
  eCoupled VL140
  eCoupled VL1000
```

Figure 6.4 The DEVS model

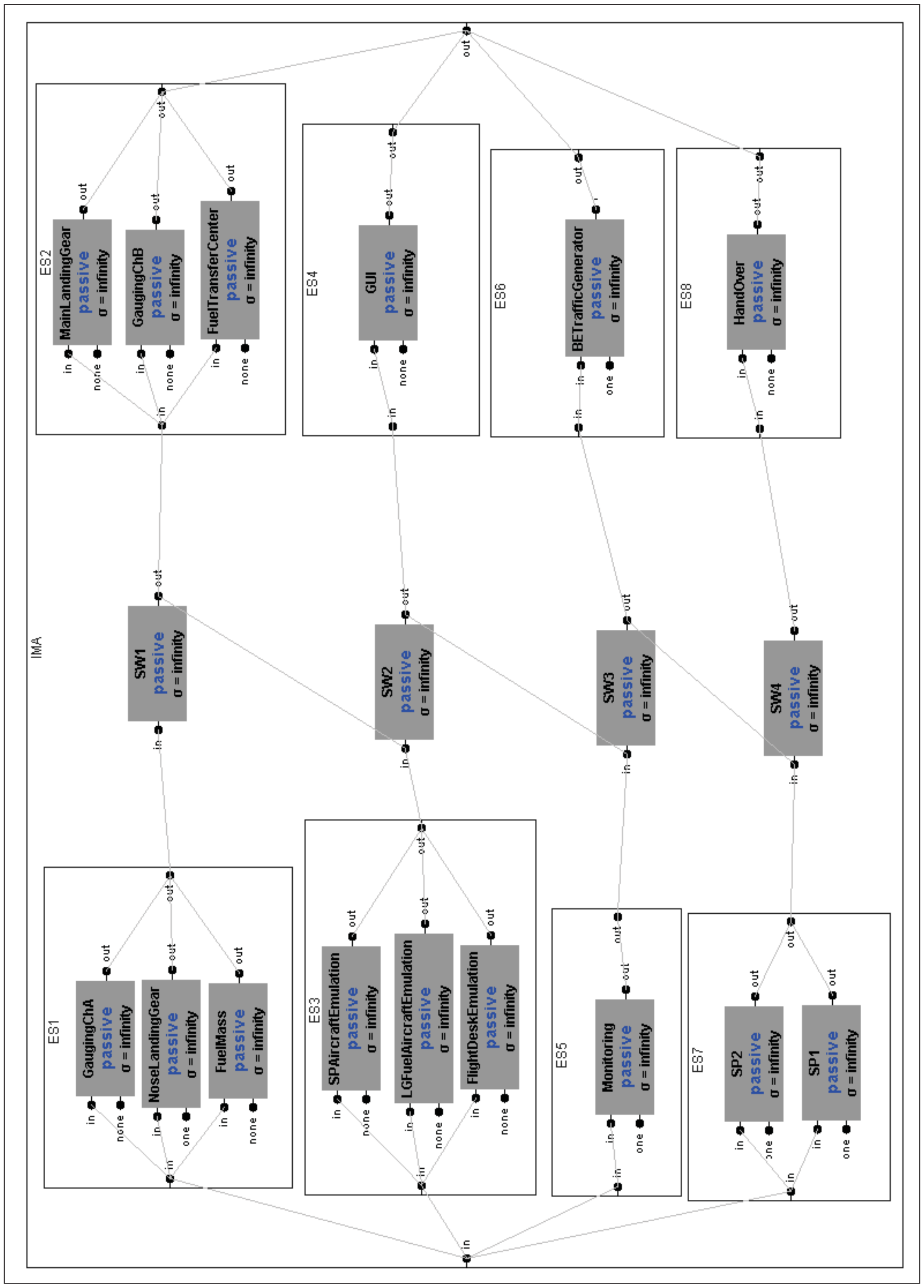


Figure 6.5 Simulation graph for Case study

CONCLUSION AND RECOMMENDATIONS

IMA architecture interconnected by TTEthernet can provide a strong platform capable of supporting mixed critical applications. The integration of these systems is challenging and costly. To address this, in this thesis we proposed and implemented a model-based technique using the Model Driven Engineering (MDE) approach for TTEthernet domain. As shown, this approach provides a metamodel to support the SAE TTEthernet standard AS6802 Aerospace (2011d) for distributed architectures through which safety-critical applications can be deployed. Based on our literature review no other attempts to integrate IMA architecture with TTEthernet have been reported. This thesis has addressed this research gap.

To begin with, we presented the metamodel of TTEthernet that captures the main concepts and characteristics of the SAE TTEthernet standard. Then we extended the standard architecture and analysis modeling language AADL to enable the modeling of integrated mixed-critical avionic applications deployed on TTEthernet-based IMA architectures. Next, for verification purposes, we selected the DEVS simulation environment, which is an environment for hierarchical and parallel formal DEVS models. We used model transformation techniques to map the metamodel of TTEthernet to the metamodel of DEVS. Then we simulated the automatically generated output model from the model transformation step in order to verify that the TTEthernet met required constraints (e.g. *contention-freedom*). The application of the proposed approach to the real-world case study yielded the following results:

- A model of the case study system generated by our TTEthernet metamodel.
- The textual syntax of the case study model generated by our AADL extension.
- An output model generated by the mapping of the case study model to the DEVS model
- An adapted output model (in this case, by adding Java code) to the DEVS simulation environment capable of verifying the *contention-freedom* property of the TTEthernet.

The main limitation of our work are divided in two parts: the limitation of TTEthernet model and DEVS verification. For TTEthernet metamodel, we have focused to be as much as possible coherent with the main concepts and characteristics of TTEthernet SAE standard. This standard such as most standards, aims at explaining the abstract concepts and features that could be used for the applications in different domains. In future we should use this metamodel to model different application and domains to understand the requirements of the next version of TTEthernet metamodel.

Currently, for DEVS verification process, the automation of the refinement step of the model transformation is challenging and still requires some significant manual input from the user to fully produce the target simulation model. As a future work, we aim at addressing this limitation. In addition, we will develop further the verification of other requirements and constraints to ensure that a system model is fully compliant with TTEthernet specification.

Briefly in future, our group will focus on developing more details of the TTEthernet metamodel in order to be capable of supporting more applications. Also, we will focus on automating the entire procedure of the approach that was presented in this thesis. However, some steps would have to be modified manually. This will make this procedure more user-friendly for system engineers; once they have produced a system model, the rest of the procedure (e.g. model transformation, interfacing with verification tools) can be automated.

In order to summarized the main benefits of this thesis, we focus on the fact of selecting TTEthernet domain and AADL modeling language. As mentioned at the outset, TTEthernet is a newly developed technology that is increasingly popular among researchers, engineers and many industries. For instance, it is currently used by NASA for reliable aerospace communication. This highlights the considerable importance of our research. However, there is a deficit of experts in this newly emerging domain that presents challenges and opportunities. The deficit presents a challenge for new researchers due to the lack of qualified experts to guide new research. This also makes it a challenge to convince industries to push development in this direction. An opportunity, by contrast, is that there are interesting possibilities for those who manage to develop expertise in this area. This is one more reason to encourage us to continue in this direction and to build on the work presented in this thesis.

AADL is selected in this thesis as a modeling language. This language has become a popular modeling language in research and industrial domains, which is due to the specific attributes of AADL that we highlight here. First of all, AADL is an open source standard; not only is the core language open source, but even all of its annexes and property sets are accessible to everyone. As previously mentioned, AADL is an extensible modeling language, which allows users to develop property sets and annexes to extend the core language. The numerous research tools that have been developed for systems analysis is another selling point of AADL. There is an SAE committee dedicated to AADL industrial and academic development, and we are a member of this committee. We present our work to them every three months for feedback from their experts. Another reason for these periodic presentations is that we are producing a networking annex in conjunction with them based on our research on the TTEthernet domain, which is also discussed in this thesis. This networking annex will be accessible for the public soon.

BIBLIOGRAPHY

"Honeywell International". http://www.honeywell.com/.

"TTTech Computertechnik AG, Wien.". http://www.tttech.com/.

A. Al Sheikh,O. Brun, P.E. Hladik. 2009. "Decision Support for Task Mapping on IMA Architecture.". (Paris, France, 2009), p. pp.31-34,. In Proceedings of the Junior Researcher Workshop on Real-Time Computing (JRWRTC 2009).

A. Al Sheikh,O. Brun, P.E. Hladik. 2010. "Partition scheduling on an IMA platform with strict periodicity and communication delays". (Toulouse, France, 2010). In proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS 2010).

Abeni, L. and G. Buttazzo. 1998. "Integrating Multimedia Applications in Hard Real-Time Systems". In *Proceedings of the IEEE Real-Time Systems Symposium*. (Washington, DC, USA, 1998), p. 4–. IEEE Computer Society.

Abuteir, M. and R. Obermaisser. July 2013. "Simulation environment for Time-Triggered Ethernet". In *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*. p. 642-648.

Aerospace, SAE. 2011a. *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex,AS5506/1*.

Aerospace, SAE. 2011b. *SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Annex B: Data Modeling Annex Annex D: Behavior Model Annex Annex F: ARINC653 Annex, AS5506/2*.

Aerospace, SAE. 2011c. *Behavioral Annex*, AS5506-X draft-2.13 ed.

Aerospace, SAE. 2011d. *Time-Triggered Ethernet*, SAE AS6802 ed.

AG., TTTech Computertechnik. 2009. *TTEthernet – A Powerful Network Solution for All Purposes*.

Ahmad Al Sheikh,Olivier Brun, Maxime Chéramy and Pierre-Emmanuel Hladik. 2013. "Optimal design of virtual links in AFDX networks". p. 308-336. Real-Time Syst.

Ahmad Al Sheikh,Olivier Brun, Pierre-Emmanuel Hladik and Balakrishna J. Prabhu. 2012. "Strictly periodic scheduling in IMA-based architectures". p. 359-386. Real-Time Syst.48.

Bartols,F.; Steinbach, T.; Korf F.; Schmidt T.C. 2011. "Performance Analysis of Time-Triggered Ether-Networks Using Off-the-Shelf-Components". Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW),14th IEEE International Symposium on.

Beji, Sofiene, Sardaouna Hamadou, Abdelouahed Gherbi and John Mullins. 2014. "SMT-Based Cost Optimization Approach for the Integration of Avionic Functions in IMA and TTEthernet Architectures". In *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*. (Washington, DC, USA, 2014), p. 165–174. IEEE Computer Society.

Bernard P. Zeigler, Herbert Praehofer and Tag Gon Kim. 2000. "Theory of Modeling and Simulation". Academic Press.

Biermann, Enrico, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer and Eduard Weiss. 2006. "Graphical definition of in-place transformations in the eclipse modeling framework". In *Model Driven Engineering Languages and Systems*, p. 425–439. Springer.

Brau, Guillaume, Jérôme Hugues and Nicolas Navet. 2013. *Refinement of aadl models using early-stage analysis methods: An avionics example*. Technical Report TR-LASSY-13-06.

Braun, Tracy D, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen and Richard F Freund. 2001. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks Onto Heterogeneous Distributed Computing Systems". *J. Parallel Distrib. Comput.*, vol. 61, n° 6, p. 810–837.

CMU/SEI. *Eclipse Modeling Framework (EMF)*. <https://eclipse.org/modeling/emf/.>.

CMU/SEI. 2014. "Open Source AADL Tool Environment (OSATEv2)". http://www.aadl.info.

De Niz, Dionisio and Peter H Feiler. 2007. "Aspects in the industry standard AADL". In *Proceedings of the 10th international workshop on Aspect-oriented modeling*. p. 15–20. ACM.

Dolange, Julien. *AADL/ARINC653 runtime POK*. <http://pok.tuxfamily.org/doku.php.id=demoarincmiddlewarequeueing>.

Domitian Tamas-Selicean, Paul Pop and Wilfried Steiner. 2012. "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems". (New York, USA, 2012), p. 473-482.

Eclipse. *Acceleo*. <http://www.eclipse.org/acceleo>.

Efftinge, S. 2006. *Xtext reference documentation*. http://www.eclipse.org/gmt/oaw/doc/4.1/r80xtextReference.pdf.

Ernesto Posse,Jean-Sébastien Bolduc, Hans Vangheluwe. 2003. "Generation of DEVS Modeling and Simulation Environment". Proceedings of the 2003 Summer Computer Simulation Conference SCSC.

F. Jouault,F. Allilaire, J. Bézivin and I. Kurtev. 2008. "ATL: A model transformation tool". p. vol. 72, no. 1–2, pp.31-39. Science of Computer Programming, Elsevier.

Feiler,Peter H.; Gluch, David P.; Hudak. 2006. "The Architecture Analysis Design Language (AADL): An Introduction". Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Frana,R.B.; Bodeveix, J.-P.; Filali M.; Rolland J.-F. 2007. "The AADL behaviour annex – experiments and roadmap". Engineering Complex Computer Systems, 12th IEEE International Conference on.

France, Robert and Bernhard Rumpe. 2007. "Model-driven Development of Complex Software: A Research Roadmap". In *2007 Future of Software Engineering*. (Washington, DC, USA, 2007), p. 37–54. IEEE Computer Society.

G. Kapos,V. Dalakas, M. Nikolaidou and D. Anagnostopoulos. "An integrated framework for automated simulation of SysML models using DEVS.". *Simulation*, p. 717-744.

Gamma, Erich, Richard Helm, Ralph Johnson and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gilles Lasnier,Laurent Pautet, Jérôme Hugues and Lutz Wrage. 2011. "An Implementation of the Behavior Annex in the AADL-Toolset Osate2". In *IEEE ICECCS*. p. 332-337. IEEE Computer Society.

Goldschmidt, Thomas, Steffen Becker and Axel Uhl. "Classification of Concrete Textual Syntax Mapping Approaches". In *Model Driven Architecture - Foundations and Applications*, Schieferdecker, Ina and Alan Hartman (Eds.), volume 5095 of *Lecture Notes in Computer Science*, p. 169-184. Springer Berlin Heidelberg. ISBN 978-3-540-69095-5. <http://dx.doi.org/10.1007/978-3-540-69100-6_12>.

Hamdane, Mohamed Elkamel, Allaoui Chaoui and Martin Strecker. 2013. "From AADL to Timed Automaton- A Verification Approach".

Inc., Wind River. 2008. "An Avionics Standard for Safe, Partitioned Systems". IEEE-CS Seminar.

Incorporated, Aeronautical Radio. 2009. *ARINC Report 664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. AEEC, Maryland, USA.

Incorporated, Aeronautical Radio. 2013. *ARINC Report 653P0 Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653*.

ISAE. *OSATE2-Ocarina*. <http://www.openaadl.org/osate-ocarina.html>.

Izosimov, Viacheslav, Paul Pop, Petru Eles and Zebo Peng. 2008. "Scheduling of Fault-tolerant Embedded Systems with Soft and Hard Timing Constraints". In *Proceedings of the Conference on Design, Automation and Test in Europe*. (New York, NY, USA, 2008), p. 915–920. ACM.

Jouault, Frédéric, Freddy Allilaire, Jean Bézivin and Ivan Kurtev. 2008. "ATL: A Model Transformation Tool". *Sci. Comput. Program.*, vol. 72, n° 1-2, p. 31–39.

Julien Delange,Laurent Pautet, Alain Plantec Mickaël Kerboeuf Frank Singhoff and Fabrice Kordon. 2009. "Validate, simulate, and implement ARINC653 systems using the AADL". In *ACM SIGAda International Conference on Ada*. p. 31-44. ACM.

Klingstam, Pär and Per Gullander. 1999. "Overview of simulation tools for computer-aided production engineering". *Computers in Industry*, vol. 38, n° 2, p. 173 - 186.

Kopetz,Hermann; Bauer, G. 2003. "The time-triggered architecture". Proceedings of the IEEE.

Kopetz, Hermann and Günther Bauer. 2003. "The time-triggered architecture". *Proceedings of the IEEE*, vol. 91, n° 1, p. 112–126.

Krahn, Holger, Bernhard Rumpe and Steven Völkel. 2007. "Integrated Definition of Abstract and Concrete Syntax for Textual Languages". In *Model Driven Engineering Languages and Systems*, Engels, Gregor, Bill Opdyke, DouglasC. Schmidt and Frank Weil (Eds.), volume 4735 of *Lecture Notes in Computer Science*, p. 286-300. Springer Berlin Heidelberg. ISBN 978-3-540-75208-0. <http://dx.doi.org/10.1007/978-3-540-75209-7_20>.

Lafaye,M.; Gatti, M.; Faura D.; Pautet L. 2010. "Model driven early exploration of IMA execution platform". Digital Avionics Systems Conference (DASC), IEEE/AIAA 30th.

Lafaye, Michaël, David Faura, Marc Gatti and Laurent Pautet. 2010. "A new modeling approach for IMA platform early validation". In *Proceedings of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. p. 17–20. ACM.

Lasnier,G.; Pautet, L.; Hugues J.; Wrage L. 2011. "An Implementation of the Behavior Annex in the AADL-Toolset Osate2". In *Engineering of Complex Computer Systems (ICECCS)*. 16th IEEE International Conference on.

Lasnier,G.; Robert, T.; Pautet L.; Kordon F. 2010. "Architectural and Behavioral Modeling with AADL for Fault Tolerant Embedded Systems". Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC),13th IEEE International Symposium on.

Lauer,M.; Ermont, J.; Boniol F.; Pagetti C. 2010a. "Analyzing end-to-end functional delays on an IMA platform". (Berlin, Heidelberg, 2010). In Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part I (ISoLA'10), Tiziana Margaria and Bernhard Steffen (Eds.).

Lauer,M.; Ermont, J.; Boniol F.; Pagetti C. 2011a. "Worst Case Temporal Consistency in Integrated Modular Avionics Systems". High-Assurance Systems Engineering (HASE), IEEE 13th International Symposium on.

Lauer,M.; Ermont, J.; Boniol F.; Pagetti C. 2011b. "Latency and freshness analysis on IMA systems". Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on.

Lauer,M.; Ermont, J.; Boniol F.; Pagetti C. Claire Pagetti;ric Boniol. 2010b. "Analyzing end-to-end functional delays on an IMA platform". (Berlin, Heidelberg, 2010), p. 243-257. In Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation.

Lauer,M.; Mullins, J.; Yeddes M. 2013. "Cost Optimization Strategy for Iterative Integration of Multi-critical Functions in IMA and TTEthernet Architecture". (Washington, DC, USA, 2013), p. 139-144. In Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual.

Liu, Hong and David P. Gluch. 2009. "Formal verification of AADL behavior models: a feasibility investigation". (New York, NY, USA, 2009), p. 6 pages. In Proceedings of the 47th Annual Southeast Regional Conference (ACM-SE 47). ACM.

LYSIC Team, University of Brest. *Cheddar Scheduling tool*. <http://beru.univ-brest.fr/~singhoff/cheddar/>.

M., José L. Risco-Martına Saurabh Mittalb Bernard P. Zeiglerb. Jesús. 2007. "From UML State Charts to DEVS State Machines using XML.". *Elsevier Science B. V.*

M. Chkouri,A. Robert, M. Bozga and J. Sifaksi. 2008. "Translating AADL into BIP - application to the verification or real-time systems". Models in Software Engineering, Springer-Verlag Berlin, Heidelberg.

Mens, Tom and Pieter Van Gorp. 2006. "A Taxonomy of Model Transformation". *Electron. Notes Theor. Comput. Sci.*, vol. 152, p. 125–142.

Michaël Lafaye,David Faura, Marc Gatti and Laurent Pautet. 2010. "A new modeling approach for IMA platform early validation". (New York, NY, USA, 2010), p. 17-20. In Proceedings of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES '10).

M.Lauer. 2012. "Une méthode globale pour la vérification d'exigences temps réel-Application á l'Avionique Modular Intégrée". Thèse de Doctorat, Institut National Polytechnique de Toulouse.

Obermaisser, Roman. 2004. *Event-triggered and time-triggered control paradigms*. Springer.

Ölveczky, Peter Csaba, Artur Boronat and José Meseguer. 2010. "Formal Semantics and Analysis of Behavioral AADL Models in Real-time Maude". In *Proceedings of the 12th IFIP WG 6.1 International Conference and 30th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Systems*. (Berlin, Heidelberg, 2010), p. 47–62. Springer-Verlag.

Pi, Zhibin Yang; Kai Hu; Dianfu Ma; Lei. 2009. "Towards a formal semantics for the AADL behavior annex". Design, Automation Test in Europe Conference Exhibition.

Robati, Tiyam, Amine El Kouhen and Abdelouahed Gherbi. 2014. "Flight Management Subsystem Model using AADL TTEthernet Extension". http://profs.etsmtl.ca/agherbi/ima.aadl.

2012. *The SAE Architecture Analysis Design Language (AADL), AS5506B*. SAE Aerospace.

Sarjoughian, Hessam S. and Abbas Mahmoodi Markid. 2012. "EMF-DEVS modeling". (San Diego, CA, USA, 2012), p. 8 pages.

Spitzer, C. 2001. *The avionics handbook*. CRCPress.

Steinbach, Till, Hermand Dieumo Kenfack, Franz Korf and Thomas C. Schmidt. 2011. "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy". In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. (ICST, Brussels, Belgium, Belgium, 2011), p. 375–382. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Steiner, W. 2010. "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks". Real-Time Systems Symposium (RTSS), IEEE 31st.

Steinhammer,K., Ademaj A. 2007. "Hardware Implementation of the Time-Triggered Ethernet Controller". *IESS 2007, pp. 325–338*.

Team, SEI AADL et al. 2006. "An extensible open source AADL tool environment (OSATE)". *Software Engineering Institute*.

Tiyam Robati,Amine El Kouhen, Abdelouahed Gherbi Sardaouna Hamadou John Mullins. 2014. "An Extension for AADL to Model Mixed-Criticality Avionic Systems Deployed on IMA architectures with TTEthernet". (Valencia, Spain, 2014), p. 14 pages.

University, Arizona. *DEVS-Suite Simulator*. <http://devs-suitesim.sf.net>.

http://www.real-time-ethernet.de. *Real-Time Ethernet in Industry Automation, 2004*.

Vangheluwe, H. L. M. 2000. "DEVS as a common denominator for multi-formalism hybrid systems modelling". In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*. p. 129-134.

Watkins, C. B. and R. Walter. 2007. "Transitioning from federated avionics architectures to Integrated Modular Avionics". In *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*. p. 2.A.1-1-2.A.1-10.

Wilfried Steiner,G. Bauer, Brendan Hall Michael Paulitsch and Srivatsan Varadarajan. 2009. "TTEthernet Dataflow Concept". (Washington, DC, USA, 2009). Network Computing and Applications, NCA 2009. Eighth IEEE International Symposium on.

Y. Lei,W. Wang, Q. Li and Y. Zhu. "A transformation model from DEVS to SMP2 based on MDA.". *Simulation Modelling Practice and Theory,*, vol. 17, n° 10, p. 1690–1709.

Zeigler., Bernard P. 1984. "Multifacetted Modelling and Discrete Event Simulation". Academic Press.

Zhang, Zhenkai and Xenofon Koutsoukos, 2013. *Embedded Systems: Design, Analysis and Verification: 4th IFIP TC 10 International Embedded Systems Symposium, IESS 2013, Paderborn, Germany, June 17-19, 2013. Proceedings*, chapter Modeling Time-Triggered Ethernet in SystemC/TLM for Virtual Prototyping of Cyber-Physical Systems, p. 318–330. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-38853-8. doi: 10.1007/978-3-642-38853-8_29. <http://dx.doi.org/10.1007/978-3-642-38853-8_29>.