FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

# TTEthernet and Modeling Languages: Introducing Methodologies

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Technische Informatik**

eingereicht von

**Valentin Ecker**
Matrikelnummer 0426030

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Puschner
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Christian El-Salloum

Wien, TT.MM.JJJJ

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

# TTEthernet and Modeling Languages: Introducing Methodologies

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Master of Science**

in

**Computer Science**

by

**Valentin Ecker**

Registration Number 0426030

to the Faculty of Informatics
at the Vienna University of Technology

| | |
|---|---|
| Advisor: | Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Puschner |
| Assistance: | Projektass. Dipl.-Ing. Dr.techn. Christian El-Salloum |

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

# Erklärung zur Verfassung der Arbeit

Valentin Ecker
Diehlgasse 9/20, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

| | |
|---|---|
| (Ort, Datum) | (Unterschrift Verfasser) |

# Acknowledgements

This work is dedicated to all the people I love! ♥

# Abstract

Time Triggered Ethernet is an open platform, which enables the use of COTS Ethernet products together with real-time segments. To facilitate the use of TTEthernet in highly dependable systems, it is important that the platform is properly supported by a modeling language.

In order to derive the special needs of a modeling language supporting TTEthernet, a study of existing modeling languages will be done and the inherent properties, constraints and possible applications of TTEthernet will be investigated. Possible use cases, variations and extensions of different modeling languages will be compared and discussed in perspective of the practical use with TTEthernet.

Further, a reference implementation will be described to demonstrate the certain benefits of using modeling languages when designing a project incorporating TTEthernet. Required extensions to existing modeling languages will be discussed. Finally, advantages of the new methodologies will be identified.

Current state-of-the-art modeling languages for safety critical embedded systems include UML and its extension UML Marte, the SAE Architecture Analysis and Design Language (AADL), SCADE (Esterel Technologies), Simulink (The MathWorks) as well as various combinations. None of the existing modeling languages has been specifically designed for TTEthernet. Existing gaps and missing features in these modeling languages concerning the design of projects with TTEthernet will be identified and evaluated.

# Kurzfassung

Time Triggered Ethernet ist eine offene Plattform welche es ermöglicht ein deterministisches und echtzeitfähiges Netzwerk zu erstellen, und gleichzeitig auch den Einsatz von COTS (Component Of The Shelf, “Produkte von der Stange” ) Produkten unterstützt. Um die Entwicklung von höchst zuverlässigen verteilten Systemen zu ermöglichen ist es wichtig, dass die Plattform von einer Modellierungssprache unterstützt wird.

Um die speziellen Anforderungen dieser Modellierungssprache zu ermitteln wird eine Studie über vorhandene Sprachen und deren Eigenschaften durchgeführt. Hierbei werden inherente Eigenschaften, Einschränkungen und mögliche Anwendungen gegenübergestellt. Mögliche Anwendungsgebiete, Variationen und Erweiterungen von unterschiedlichen Modellierungssprachen werden in Hinbick auf die praktische Verwendung mit TTEthernet verglichen und diskutiert.

Weiters wird eine Referenzimplementierung entwickelt um die Möglichkeiten und Vorteile von Modellierungssprachen in Verbindung mit TTEthernet aufzuzeigen. Benötigte Erweiterungen zu exisiterenden Sprachen werden implementiert und diskutiert. Anschliessend werden die Vorteile dieser neuen Methode aufgezeigt.

Aktuelle state-of-the-art Modellierungssprachen für sicherheitskritische Echtzeitsysteme umfassen UML und dessen Erweiterung UML Marte und SysML, AADL (SAE Architecture Analysis and Design Language), SCADE (Esterel Technologies), Simulink (The MathWorks) und verschiedenste Kombinationen. Da keine dieser Sprachen speziell für TTEthernet geschrieben wurde, werden existierende Diskrepanzen und fehlende Eigenschaften in Bezug auf TTEthernet ermittelt und bewertet.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation and problem statement

TTEthernet is an open platform which enables the use of COTS Ethernet products together with real-time segments. To facilitate the use of TTEthernet in highly dependable systems, it is important that the platform is properly supported by a modeling language. Currently there is no state-of-the-art methodology which facilitates the design and integration of projects incorporating TTEthernet.

The reasons why it is important to have the support of a modeling language methodology are:

- The use of a methodology supports the assessment of feasibility already at the beginning of a project due to a structured approach.
- A methodology helps with project planning, specifying, constructing and documenting.
- It simplifies the traceability of requirements.
- It facilitates easier communication between developer and project management due to a common base of "technical" language.
- The use of a methodology simplifies documentation, which is more consistent with the current coding status.

This are only some examples that evolve when designing projects using a modeling language, which explains why it is not only common but more and more obligatory to use a methodology when realizing complex projects as done in the fields of safety critical real time systems.

## 1.2 Aim of the work

The aim of this thesis is the development of a methodology for designing TTEthernet applications using modeling languages. A reference implementation will be developed including necessary extensions for existing modeling languages.

## 1.3 Methodological approach

In order to derive the special needs of a modeling language supporting TTEthernet, a study of existing modeling languages will be done, and inherent properties, constraints and possible applications of TTEthernet [1] will be investigated. Various use cases, variations and extensions of different modeling languages will be compared and discussed in perspective of the practical use with TTEthernet.

Further, a reference implementation will be described to demonstrate certain benefits of using modeling languages when designing a project incorporating TTEthernet. Required extensions to existing modeling languages will be implemented and discussed, subsequently an experimental validation will be done. Finally, advantages of the new methodology will be identified.

# Chapter 2

# Study of existing Modeling Languages

## 2.1 UML Marte

The Unified Modeling Language is a well known language used for modeling software projects with an object oriented focus. In this domain the language reached its version 2.3 in 2010 and spread widely in the last decade. However, because its focus lies on the modeling of large object oriented projects, it lacks capabilities for the modeling of safety critical applications. Mainly because of its lacking possiblity to incorporate the time domain. To overcome these problems, an attempt has been made in form of a profile. In 2009, the Object Management Group (OMG) adopted the UML Marte profile for UML to properly support the design of real time systems. It succeeds and extends the SPT profile (UML Profile for Schedulability, Performance and Time) which had some practical shortcomings in terms of expressive power and flexibility [2].

### Extensions to UML

The time model (see figure 2.1, [3]) UML Marte uses, allows very diverse time modeling since it not only relies on physical time, but also on logical time. Multiple time bases are also allowed, with the possibility to relate them to each other, as well as the possibility to relate single instances to each other to represent partial orders. Since the time model in UML Marte is merely a mathematical model, clocks are implementations of time bases and give access to the time structure, where e.g. a *chronometric clock* describes a physical clock. Also the time structure relations can be defined in a formal way. For that, a non normative *Constraint Specification Language* was specified, since the *Object Constraint Language* (encouraged by UML) was not suited for this task. To further give the possibility to define the behaviour of time triggered architectures, the UML Marte stereotypes *TimedEvent* (which extends the UML metaclass *TimedProcessing*) and *TimedProsessing* (which extends the metaclasses *Action*, *Behaviour* and *Message*) were introduced. These metaclasses can be bound to clocks.

The whole time structure is heavily inspired by the Tagged Signal Model [4].

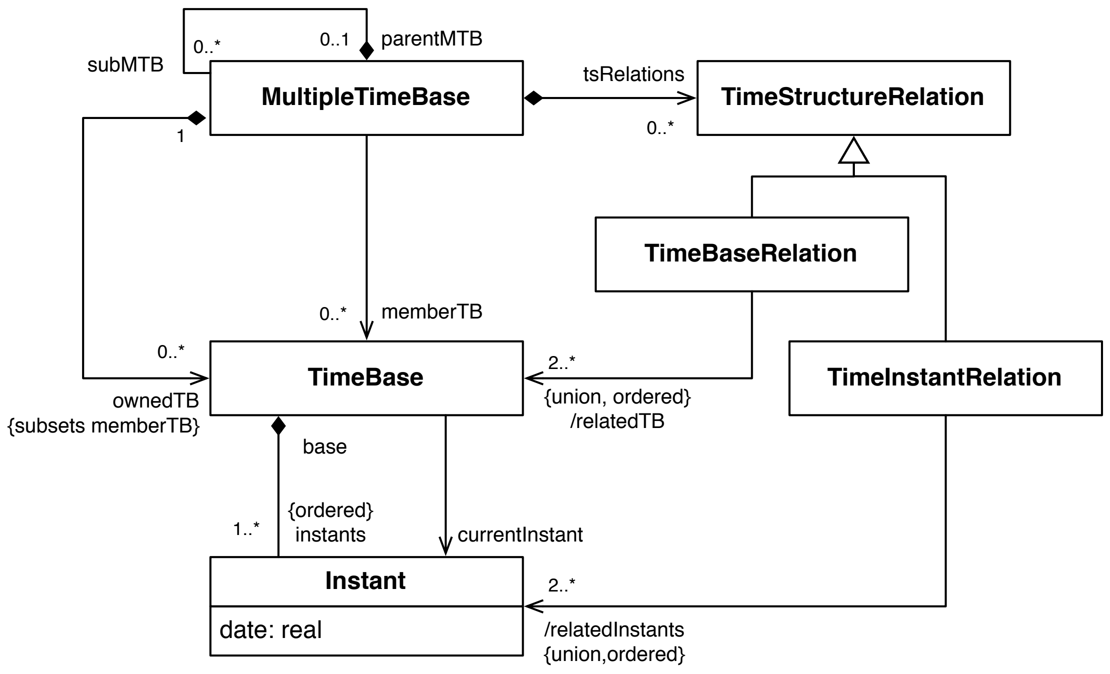


**Figure 2.1:** The time structure of UML Marte, [3]

The second topic UML Marte defines to help designing real time systems, is the allocation model [3]. Where in UML Marte allocation covers both temporal scheduling and spatial distribution, divided into *application* and *execution platform*. The allocation model then connects these two submodels in the sense of providing constraints for the clocks and instants to which they are connected to. An allocation can further be abstract, behavioural or hybrid.

## 2.2 SysML

*» SysML supports the specification, analysis, design, verification, and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities. «* [5]

In contrast to other modeling languages, SysML is more oriented to model real world applications than specific embedded systems. It was derived from UML and uses it as basis, but dismisses all features that are only needed when designing software projects. Further, it alternates diagram types where there is need for a more general representation. It can be said that the language tries to be as general and unspecific as possible, but nevertheless giving powerful constructs to design in detail as well. Furthermore it aims to replace and unifies all the different modeling languages that are currently used by a system engineer. Similar to what UML did in the software domain.

Historically, SysML is a project that evolved from the demand for a more general modeling language than UML with its software centric construction, but yet using the well known syntax and already well established knowledge of UML. With the start in 2001 and after some workgroup disagreements in 2005, SysML was finally accepted by the Object Management Group in 2006. (see [6]). It is possible that SysML will be customized to accommodate certain specific domains, such as Aerospace or Automotive.

What is new in SysML and what are the key differences to UML?

- Classes from UML are called System Blocks to emphasize the more general use.
- Information Flow between elements can be modeled.
- The Requirement Diagram and Parametric Diagram have been added.
- The Activity Diagram, Block Definition Diagram and the Internal Block Diagram have been adapted.
- Some Diagrams that were to software-centered have been removed to keep the standard as lightweight as possible.

The necessity of the adaption of UML becomes clear when thought of an example, like a subway train. In UML, definitions of requirements are only covered by use cases. In SysML which added the Requirement Diagram, there is now the chance to define hard requirements ( e.g. the minimum and maximum speed or the acceleration) in a clear and unambiguous way. Also, the requirement exchange with other connected and/or interdependent systems becomes easier. Think of the supply voltage of the underground train, which has to stick to certain boundaries seen from the power grid, and has to be accepted in certain boundaries seen from the train side.

SysML Diagrams consist of the already mentioned Requirement Diagram, the Behaviour Diagram and the Structure Diagram. As the name already suggests, the overall system structure is defined by the Block Definition Diagram and the Internal Block Diagram, which are both Structure Diagrams. Organizing is satisfied by the Package Diagram. The Behaviour Diagrams add high level descriptions with the Use Case Diagram, as well as in-depth views with the State, Sequence and Activity Diagrams, which describe the behaviour of the model in a fine grained way. The Activity Diagram shows control and data flow between activities, the Sequence Diagram the interaction related parts, and the State Diagram gives a view of internal states and their transition conditions. (see [7] for further details)

Figure 2.2 shows the SysML taxonomy, and Figure 2.3 the UML taxonomy as reference.

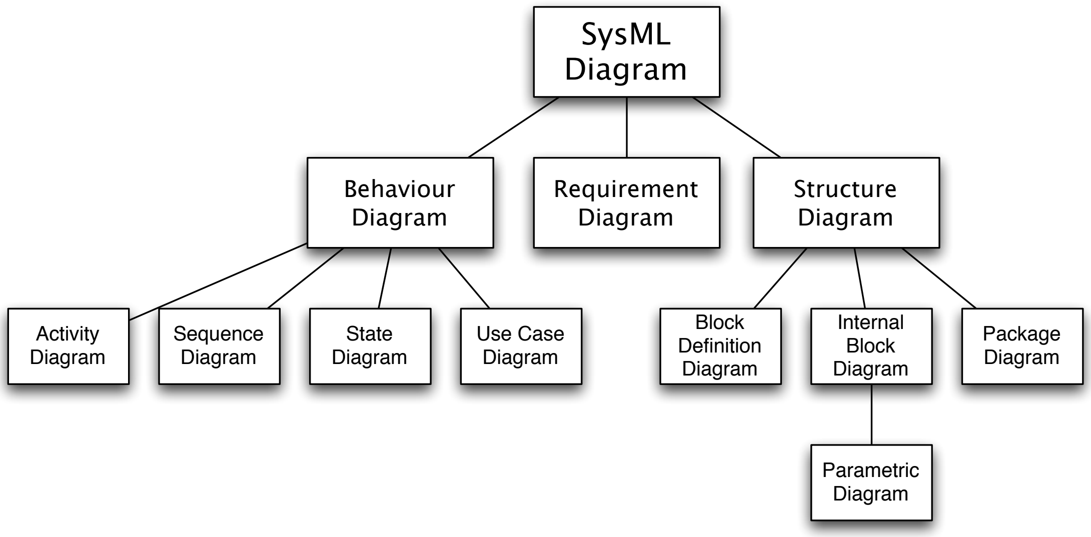


**Figure 2.2:** SysML Taxonomy

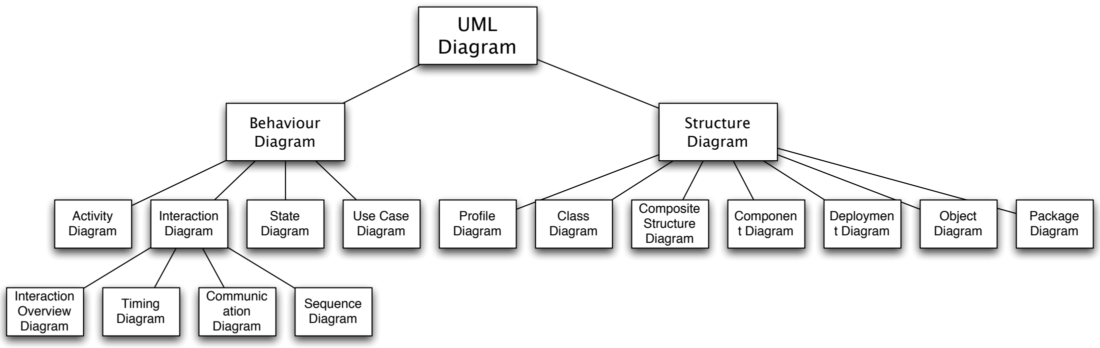


**Figure 2.3:** UML Taxonomy

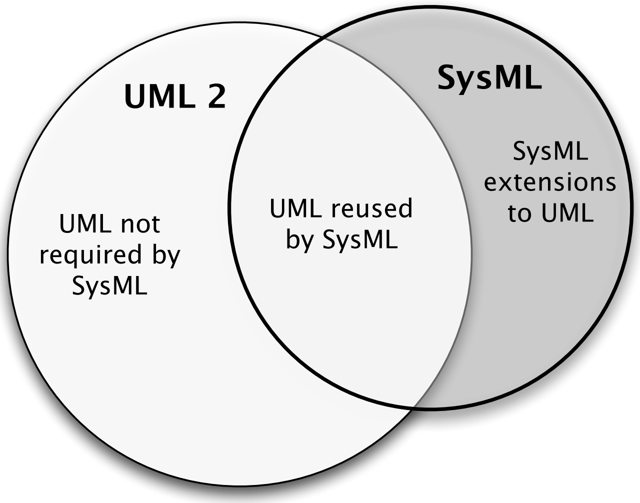


**Figure 2.4:** SysML UML inheritance

## 2.3 SCADE

SCADE (Safety Critical Application Development Environment) is a toolset for systems modeling as well as model based software development. It is a commercial product from Esterel Technologies. Programming in SCADE is primarily graphical but based on the Lustre programming language [8], which is flow-based, declarative and synchronous. It mainly addresses the user part of a software solution for hard real time application [9]. Therefore all other parts, like Operating System, Drivers and Hardware must be supplied and have to be certified as well if needed. See Figure 2.5.

In a synchronous programming language (or synchronous reactive programming language), the paradigm is, that the computer program reacts to inputs given in a certain interval. The sampling and further the output interval must meet a-priori defined timing constraints to fulfill certain real world constrains. This model is a direct implementation of the sampling-actuating model used in the field of control engineering, which makes it very attractive for engineers acting in this business.

### Lustre

Lustre is the language behind SCADE and was developed as a synchronous programming language for reactive systems. It's declarative, flow based and a time based approach, which makes the language predestined for safety critical and reactive applications. The data flow aspect is maybe one of the most distinguishable features, because it gives somewhat of a "natural" perception of how data is handled and modified. This keeps the language simple and comprehen-

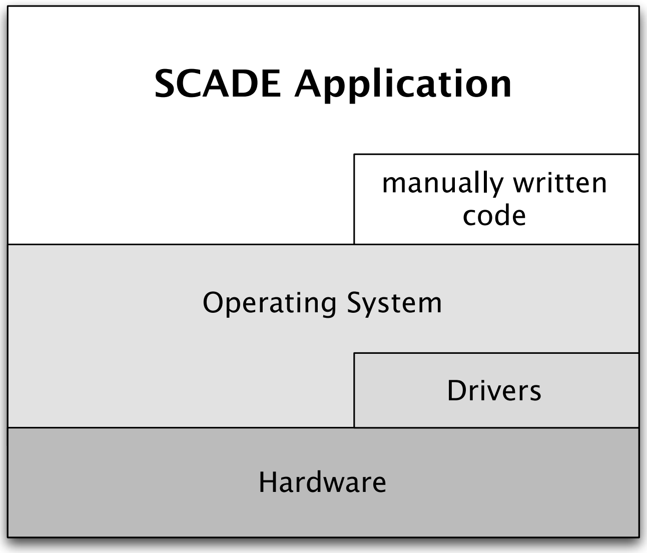
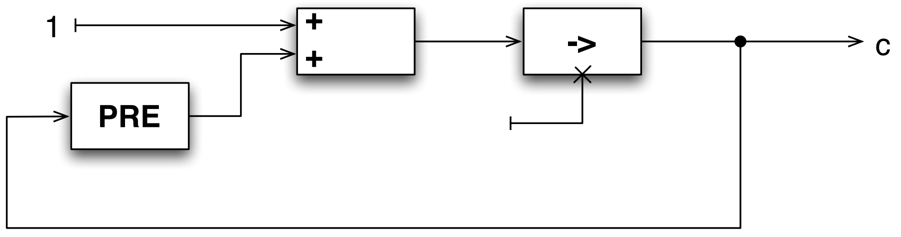


**Figure 2.5:** SCADE target, compare [9]



c = 0 -> (1 + pre c)

**Figure 2.6:** A simple counter in SCADE

sible, a very desirable attribute in this field of system design. Further, the inherent handling of time makes it easier to verify the timing behaviour of the model, helping to cope with hard dead lines imposed from the real world interface.

Figure 2.6 is a short example of a simple counter programmed in SCADE as described in the paper *Scade 6: a model based solution for safety critical software development* [9]. It shows the basic modeling principle, and the close connection to Lustre as well. The *PRE* operator is similar to a memory that stores the last value given from its input. Correct initialisation has to be taken care of. The -> operator ensures the mentioned initialisation with zero in our example.

### SCADE and UML

Since describing a model in SCADE is only possible in the behavioural domain, it is obvious that other modeling languages are used for covering the higher levels of the model, such as requirements. Research [10] was already done with focus on using UML for high level modeling, while still using SCADE for behavioural description and code generation. It was shown that the block interface in UML can be mapped to a block interface in SCADE. In certain cases the mapping can be automated as well, which gives the advantage of lesser maintaining while designing at the beginning, and – more important – in the later stages of the project status, where changes are more difficult to manage. Hybrid solutions are also possible, where some blocks get mapped to the SCADE model, and some can be kept in the UML domain without changing the paradigm.

Since UML is only used for defining asynchronous message passing and signal transition, and SCADE is bound to the synchronous paradigm, some "bridging" has to be made. For this, a *signal presence status* must be added to every data flow. This type can be represented by a boolean value, either as an stand-alone data flow, or packed in a tuple. This signal ensures a symmetrical triggering of state machines both in UML as well as in SCADE. The same technique holds for output data flows, where the *signal presence status* is only set true *iff* a signal is emitted during execution.

Further, diverse research also developed methodologies for the node and interface mapping as well as hierarchy and multiplicity handling [10].

### SCADE and Simulink

Simulink is heavy used for designing and simulating in the domain of control design, but is clearly not designed for code generation in the embedded domain as it will be proved in a later section. To fill this gap, SCADE could be used as a lower level modeling tool to program the behaviour specified and simulated in Simulink. Research [11] on this topic has been done to give an end-to-end approach from Simulink via SCADE down to the Time Triggered Architecture [12].

To convert a Simulink model to a SCADE/Lustre model, several limitations have to be considered and several transformations have to be made. It can be noted that these two languages have very much in common, e.g. the data-flow language, or the block semantics. But there are also some strong differences [11]:

- *Typing*: SCADE is strongly typed, whereas in Simulink explicit types are not mandatory.
- *Semantics*: In SCADE they are precise and unique, in Simulink they depend on the simulation method.
- *Time*: Simulink allows only modeling of continuous time models, where even signals in the discrete-time library are only piecewise constant continuous signals. SCADE only allows discrete models.
- *Modularity*: Simulink allows non-modularity, e.g. it is possible to run a system A, which is inherited by system B with a higher clock rate as system B. This is not possible in SCADE.

SCADE is already used in many safety critical applications like helicopters, planes, railways or nuclear power plants, enabling control and feedback control systems, graphical displays, navigation etc.. Because it is well tested in many different fields and practically an industry standard in the niche market of dependable systems, SCADE became one of the mostly used toolset for model based development in this area. Especially with various gateways and automatic translation to other design tools such as The Mathworks Simulink or UML it is possible that SCADE will further extend its role in this niche. One downside of course is that the whole "package" of tools, support and experience is provided only by Esterel at the moment. Therefore a certain amount of dependency has to be accepted if this toolset is chosen.

## 2.4 The Mathworks Simulink

Simulink is a commercial tool for modeling, simulating and code generation. It comes as an add-on to Matlab, which is primarily used for numerical computing. Simulink has a long history and is now well used when it comes to modeling of control theory tasks or physical domains (see an example in figure 2.7). Certain toolboxes enable the modeling and simulation of communication systems, aerospace components or even neural networks. Stateflow allows to model finite state

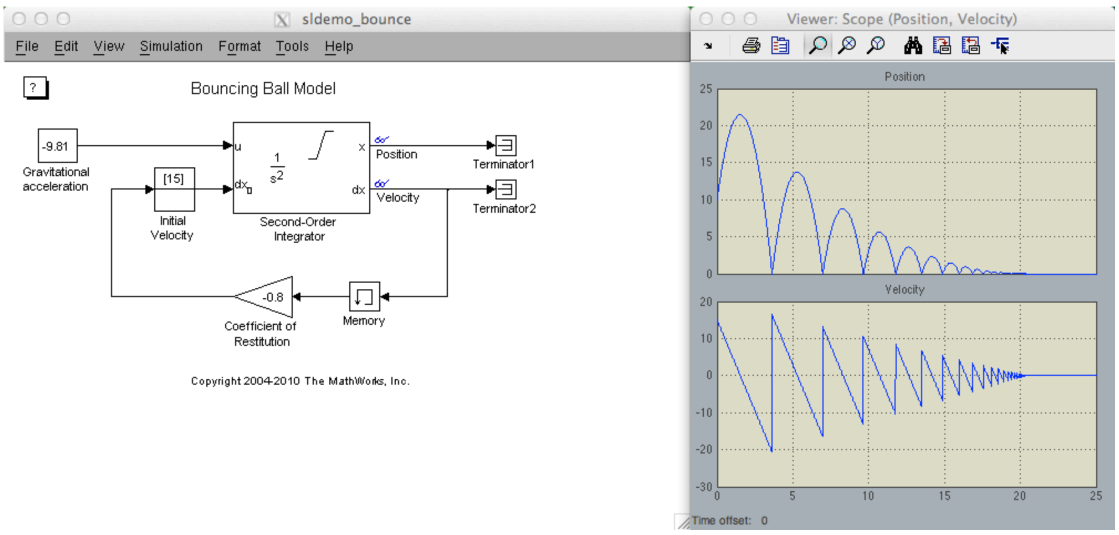

sldemo_bounce
File Edit View Simulation Format Tools Help
Bouncing Ball Model
-9.81
Gravitational acceleration
[15]
Initial Velocity
Second-Order Integrator
Position
Velocity
Terminator1
Terminator2
-0.8
Coefficient of Restitution
Memory
Copyright 2004-2010 The MathWorks, Inc.
Viewer: Scope (Position, Velocity)
Position
Velocity
Time offset: 0

**Figure 2.7:** A simple model of a bouncing ball in Simulink

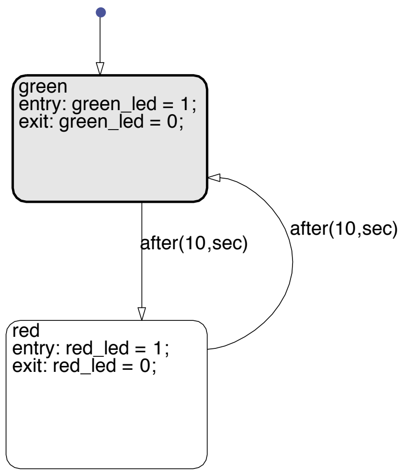


**Figure 2.8:** A simple model of a pedestrian traffic light in Stateflow

machines (see figure 2.8), which is an interesting capability for the purpose of modeling safety critical applications.

Matlab and Simulink along with various toolsets are well used and tested when it comes to modeling and simulation. Various extensions exist to automatically generate C code or VHDL code from a Simulink model, as well as for rapid prototyping on embedded hardware. On the downside, the existing code generators do not guarantee that the behaviour of the generated code is the same as the one modeled [13]. Thus, they are not suitable for the safety critical domain, but translators to other languages – which enable automatic and verified code generation

– exist [13]. But it should be noted that these transformations come with some drawbacks in terms of restricted modeling and simulation possibilities (see section 2.3).

## 2.5 SAE Architecture Analysis and Design Language (AADL)

AADL is a language mainly used in and specified by the aviation and automotive industry. It enables detailed and unambiguous system modeling of architectures typical in this domain, as e.g. distributed, fault tolerant embedded systems. The language is text based, but can be interpreted by a graphical editor as well. Similar to other modeling languages in this domain, AADL enables a component based description of virtual and physical components and resources. The data exchange paradigm is based on data flows – which in general can be compared with the ones in Simulink or SCADE– but also includes access to shared memory or remote procedure calls. Since AADL mainly targets embedded applications in the safety critical domain, real time concerns are also taken care of. That includes the support of deadline attributes, scheduling protocols or runtime system reconfigurations for fault-tolerant configurations. Also, like other runtime architecture modeling languages, AADL can be used in conjunction with other languages that facilitate the modeling of non-functional structures as well. The paper *Diagrams and languages for model-based software engineering of embedded systems: UML and AADL* [14] shows how AADL is compared with UML and SysML.

The component abstractions are divided in three subclasses and include [15]:

*Application software:*

- thread
- thread group
- process
- data
- subrogram

*Execution platform:*

- processor
- memory
- device
- bus

*Composite:*

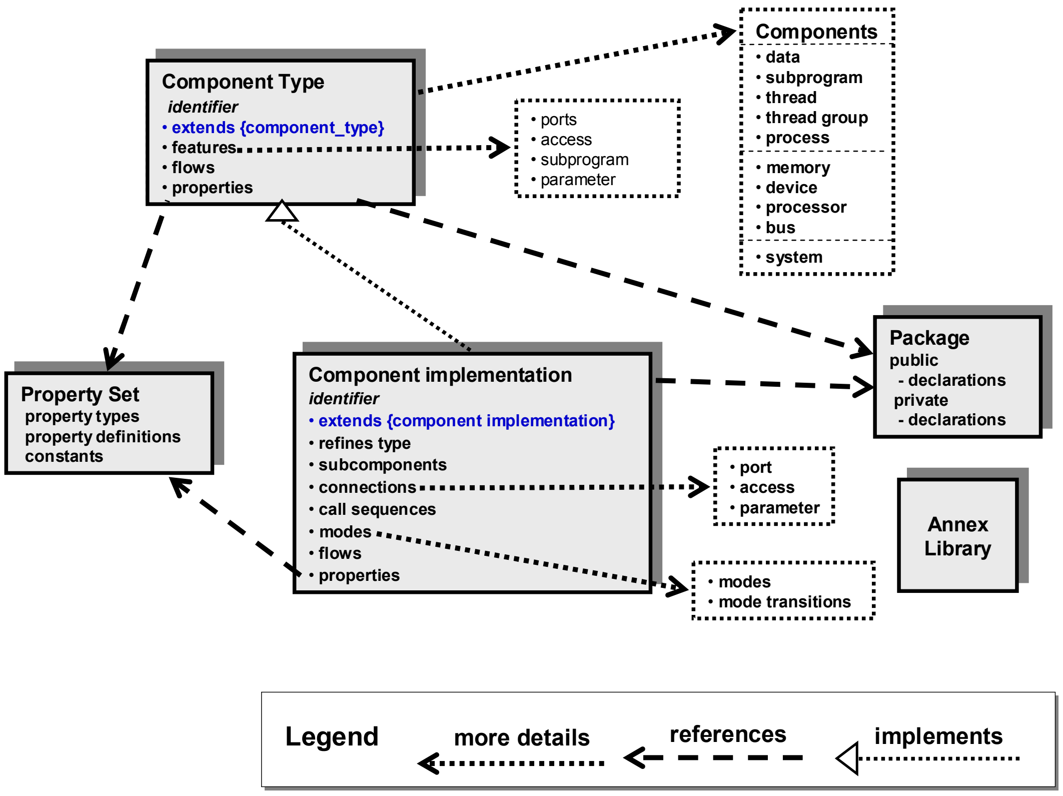


**Figure 2.9:** AADL elements summary [15]

- system

Each of these components properties can be described in detail, which can later be used for analysis. E.g. each thread can be associated with an (variable) execution time, giving architecture information that can be used for a schedulability analysis or verifying that all deadlines will be met throughout the system. Or each data size attribute can be defined to check on later if the memory given by the execution platform will be sufficient for all eventualities. AADL is primarily used for unambiguous description and static analyzing, but has also been considered for the automatic generation of Ada code using the OCARINA toolsuite [16], [17]. Another toolset, called TOPCASED [18], is used for a formal verification approach [19].

Figure 2.9 shows a summary of all AADL elements.

Due to the fact that AADL is primarily designed for the use in the embedded safety critical domain, it seems that this modeling language is perfectly suited for the design of TTEthernet. But as of today, AADL is not very wide spread in the industry, although it is a very promising approach and will very likely receive more attention in the future.

## 2.6 Giotto

Giotto [20] is a language that gives the engineer the possibility to take the concern of timing directly into the modeling task. Also, it features an abstraction of the actual platform, giving the facility to model without the concern of execution times, leaving the actual verification on keeping deadlines to the compiler.

The basic architecture of Giotto is divided in tasks, modes – which consist of one or more tasks – and drivers. One or more tasks can form a node and communicate through drivers. The programmer can define invocation rates for every single task, to make sure the a-priori defined deadlines are met. The execution of modes and tasks is strictly time triggered, where the Giotto program does not define the execution interleaving of the tasks. Compiler directives can be given to enforce certain constraints, such as the execution of task A on node A and the execution of task B on node B.

Annotated Giotto gives the possibility to further influence the execution on different platforms as well as it enables the building of redundant systems, or the allocation of certain networks. The three levels of annotations are top down, which means that the higher level annotations have to be defined, before the lower levels can be. Following definitions are cited from *Giotto: A time-triggered language for embedded programming* [20]:

- "*Giotto-H* (H for hardware) specifies a set of hosts, a set of networks, and worst-case execution time information. The WCET information includes the time needed to execute tasks on hosts, and the time needed to transfer connections on networks."
- "*Giotto-HM* (M for map) specifies, in addition, an assignment of task invocations to hosts, and an assignment of connections to networks. The same task, when invoked in different modes, may be assigned to different hosts. The mapping of a task invocation also determines the physical location of the task output ports."
- "*Giotto-HMS* (S for schedule) specifies, in addition, scheduling information for each host and network. For example, every task invocation may be assigned a priority, and every connection may be assigned a time slot. "

Giotto gives the chance to describe behavioural an temporal behaviour in detail, promising the same behaviour on different platforms. This sets this language on the same level as SCADE/Lustre.

Chapter 3

# Time Triggered Ethernet

## 3.1 Introduction

In the last decades, safety critical applications based on distributed embedded systems became one of the most needed but also most challenging topics in computer science. Because of the ever-growing demand of robustness, reliability and physical distribution of components, the need for reliable communication among those systems became immanent. Several communication protocols have been designed to fulfill that need. From simple event based message passing, using various media access controls, up to the Time Triggered Architecture [12]. Especially in highly safety critical applications, such as in aerospace, automotive or various industrial domains, it became clear that only highly tailored protocols (e.g. using time triggered communication) would fulfill all needed properties. Various protocols such as TTP/C [21], SAFEbus [22] or FlexRay [23] have been developed. Hybrid protocols such as FlexRay, AFDX [24], or TTEthernet extend the use cases by adding the possibility to communicate via asynchronous message passing while preserving the synchronous communication paradigm where needed.

TTEthernet [25] further makes it possible to use "component of the shelf" equipment for non safety critical traffic. The architecture guarantees that non-critical communication can not influence the safety critical communications. To ensure this behaviour, three different traffic classes have been defined: Best-Effort traffic, Rate-Constrained traffic and Time-Triggered traffic. As the naming suggests, Best-Effort traffic is used for communication where no constraints or guarantees on message traveling time or jitter are given. Event based communication based on standard Ethernet can be done in this class. With Rate-Constrained traffic, bandwidth is reserved for the communication of a certain application. It does not guarantee that messages can be passed at certain points in time, but overall a-priori defined bandwidth requirements will be met with bounded and defined jitter. In contrast to these two classes, Time-Triggered traffic guarantees bandwidth, as well as exact time in a communication schedule.

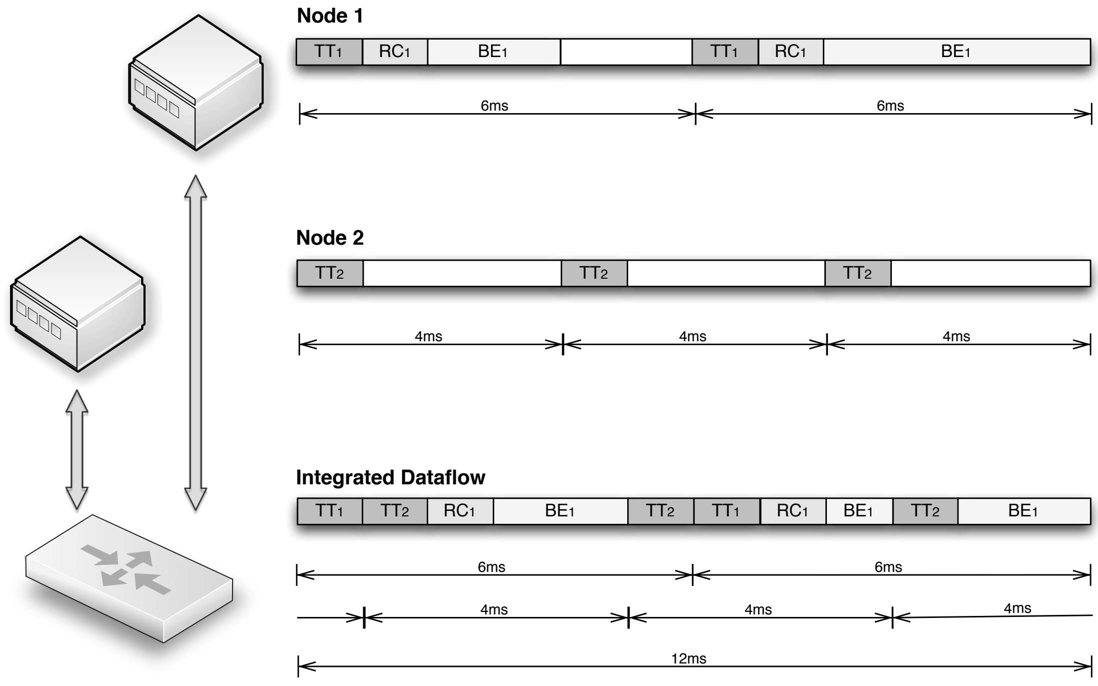


**Figure 3.1:** TTEthernet Dataflow Integration

## 3.2 TTEthernet technology

Concerning the three already discussed traffic classes, we will mainly focus on the Time-Triggered class, since this is the class with the features we are most interested in.

### Time and Traffic Classes

One of the most important aspect when talking about time-triggered communication is - as the name suggests - time. The communication in the Time-Triggered traffic class is organized in cycles (similar to TTP/C), where one cycle consist out of 0..n frames. Frames are either Rate-Constrained or Time-Triggered traffic, the remaining time can be used for Best-Effort traffic. The communication schedule repeats with every cluster cycle which integrates several cycles from different schedules.

Besides the endsystems, the TTEthernet Switches are the central unit of the network. They ensure – besides connecting the links – that all schedules of the different cycles are interleaved properly, in order to meet all traffic class properties. This mainly concerns the Time-Triggered and Rate-Constrained traffic classes, since these are the only classes that demand certain sending points in time or overall bandwidth per cycle respectively. Offline analysis has to ensure that the different cycles can be interleaved properly by the switch.

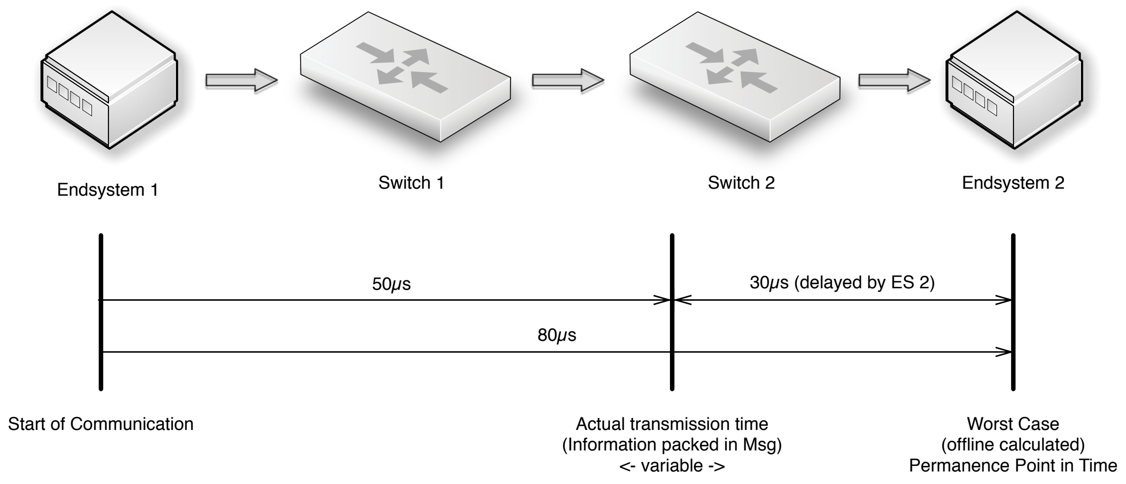


**Figure 3.2:** TTEthernet and transmission time compensation

Figure 3.1 shows an example dataflow Integration with two endsystems. Node 1 has a cycle period of 6ms and demands one slot for Time-Triggered traffic, one for Rate-Constrained traffic and two slots are left for Best-Effort traffic in the first displayed cycle, and four in the second. Node 2 has a cycle period of 4ms, but usually sends only Time-Triggered traffic with one slot. After dataflow integration we can see that Node 2 is integrated with a phase shift of one slot, guaranteeing that the Time-Triggered triggered traffic suffers no jitter. Rate-Constrained traffic will be distributed in the free slots with the guarantee that the demanded amount of slots (equals bandwidth) will be met, but bounded jitter can occur. The rest of the time can be used for regular Best-Effort traffic. It has to be noted that this is the only class where data packets can be dropped by the switch in overload scenarios. Hence, in the example, we can see the Best-Effort traffic in the second slot of Node 1 is not guaranteed to be transmitted (and dropping could occur). The cluster cycle time is the least common multiple (LCM) of all integrated cycles (12ms in our case).

Basis of this TDMA (Time Division Multiple Access) based resource sharing are local clocks that are sufficiently synchronized to each other. This is ensured by a clock synchronization protocol, which is periodically executed via Protocol Control Frames (PCF). To achieve a high precision of the clock ensemble, it is also crucial to have sufficient information about the delays in the network when executing the synchronization protocol. In TTEthernet this is ensured by the switches which add the information about the amount of imposed dynamic and static delay to the PCFs. This further allows the receiver to delay all PCFs up to the offline calculated worst case transmission delay and therefore to re-establish the send-order of all messages regardless of the network layout. The point in time when a message is given to the higher layer (when the send order is re-established) is called *Permanence Point in Time*.

Figure 3.2 shows a simple example of the re-establishing of the send order. A message is sent by Node 1 with the target Node 2 with two hops in between. As the message leaves Switch 1, the Switch adds the imposed delay on the message in a designated field in the message header.

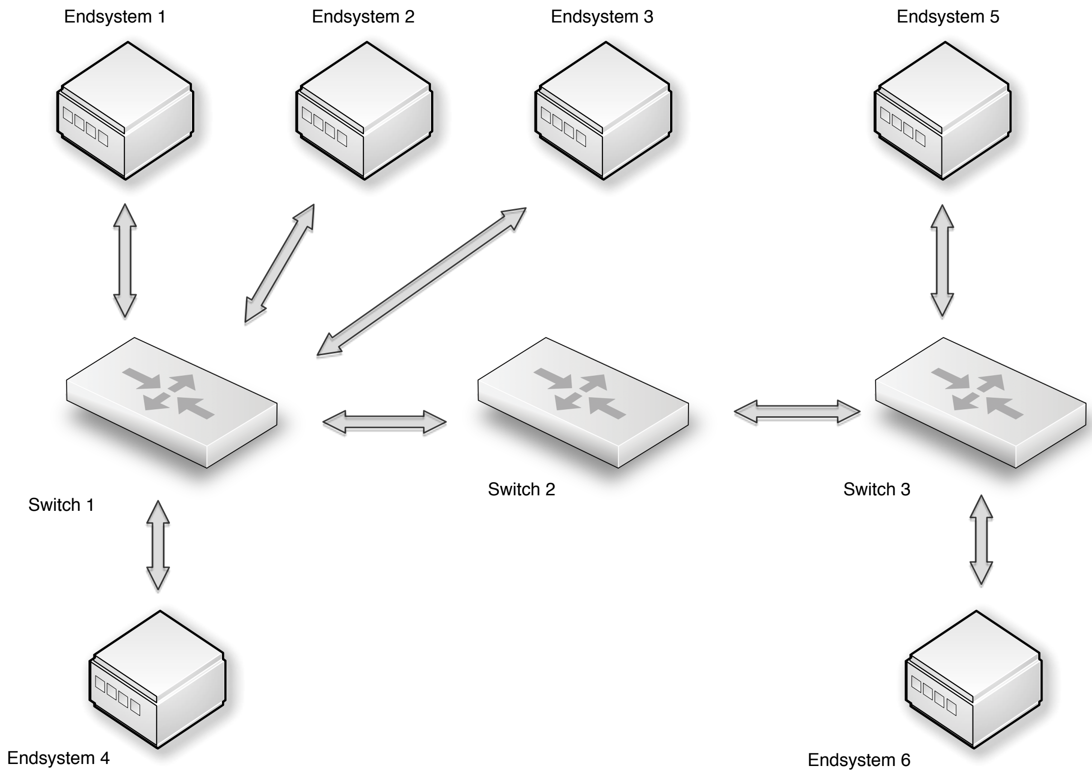


**Figure 3.3:** TTEthernet, simple Topology

Same holds for Switch 2. Upon reception in Endsystem 2, an algorithm calculates the difference between worst case travel time and actual travel time read from the message header and delays the message accordingly. It has to be noted that the message traversal delays on the physical medium also have to be calculated offline and loaded into the devices.

### Topologies

Similar to "standard" Ethernet, a TTEthernet network can consist of multiple (TTEthernet-)Endsystems and multiple TTEthernet-Switches, where an Endsystem must be connected to a Switch and Switches can be connected to each other as well. Figure 3.3 shows a simple topology with multiple switches and endsystems. But since this technology is aimed to be used in a safety critical environment, redundancy is obligatory. The notion of abstraction concerning redundancy are channels. A channel is defined as a route from the designated source to one or multiple sinks. To achieve redundancy, multiple channels via independent hardware are needed. Every redundant network does not per se have to be connected to the same nodes. E.g. it is possible that certain nodes are only connected to one network. But note that the involved channels can get restricted in terms of bandwidth and sending points in time through the schedules of the redundant traffic at the endsystems.

## 3.3 Possible applications

Control Systems Engineering can be seen as the Nr. 1 application for TTEthernet. In the last decades, Control Systems Engineering evolved from the continuous to the discrete domain. Accordingly, the same happened with communication paths. From simple analogue voltage signalling, via hard wired connections, up to message passing via digital networks. With the necessity to share communication networks in order to safe space, weight and money, the discussion of media access control came up, and with it the classification of data with respect to their meaning. TTEthernet, along with other similar communication protocols, such as FlexRay, try to satisfy all different communication scenarios such as supporting Control Systems Engineering data with high reliability and e.g. Internet surfing at the same time.

A prime example is a single airplane network which can handle Fly-by-Wire communication as well as in-flight entertainment. All at the same time with no interference on safety critical parts. The security question whether it makes sense to install a physical access to a safety critical network to every passenger seat is left open. In terms of needed bandwidth it can be said – with exceptions – that communication concerning the Control System is of low bandwith but high priority, and for in-flight entertainment demands are vice versa. Exceptions of low bandwidth Control Systems are e.g. video streams from cameras that are used for object detection or path finding. If this task is said to be mission critical or safety related, communication guarantees have to be made. Currently TTEthernet is specified and already used in networks with speeds up to 1Gbit/s.

Other use cases could be applications that can be widely described as multimedia applications. Traditionally, the more appropriate traffic classes in this section would be the Best-Effort and Rate-Constrained class of TTEthernet since usually this area is covered by *Quality-of-Service*, which has similar mechanisms. One exception could be remote video game playing. This technology consists of a small low performance receiver which is connected to a TV set and a controller for input. A remote server farm is responsible for executing and actual rendering of the video game and streams the content to the receiver in the users home. As it turned out [26], the main issue here is the transmission from the server to the receiver and vice versa in terms of packet delay and loss, resulting in a slightly delayed input to output loop with some hiccups now and then. Quality of Service (*QoS*, as seen in use with Voice over Ethernet) could reduce this effect, but does not give any guarantees on the delay, since *QoS* does rely only on prioritisation and not on network planning. But this can be seen as the main issue with TTEthernet in such areas of application. A-priori network planning is very difficult not knowing the possible number of clients and their distribution before deployment.

### Membership

Another important feature of a time triggered communication network, is the handling of membership. Since with the Time-Triggered traffic class periodic state message transmission is mandatory, it is possible to detect if a node is failing (from message omission up to byzantine faults, certain amount of nodes given [25]) in a time period known a-priori. This is usually referred to as a membership algorithm. It decides which nodes can be trusted and which not, and also guarantees that all nodes have the same view of trusted and untrusted nodes at all times. Additionally, clique detection and resolving is done to avoid the highly unpleasant and unintended scenario where two distinct logical networks are formed that do not trust each other.

# Chapter 4

# Modeling Languages and Time Triggered Ethernet

In the last chapters, a short overview of different modeling languages with their possible applications and constraints was given as well as a short overview of TTEthernet which we are interested in to model. But what might be the main difficulties that will be encountered when modeling a TTEthernet network? To answer this question, a closer look at the features of TTEthernet has to be taken. In particular, it has to be decided which features can be abstracted with a higher level and which features are influenced by the system design. This distinction is necessary since the internals of TTEthernet should not be modeled, but rather a system that utilizes TTEthernet.

## 4.1 General Issues with modeling languages

First of all, the topology of the network including all endsystems and all switches and the connections between them should be expressed. This part can either be modeled showing the physical or the logical topography. In the physical view it is possible to construct the hardware topologies including the connections and also the properties that are bound to the hardware parts like bandwidth on a connection, interconnection bandwidth in a switch as well as power or physical space demands. But if logical elements such as channels and their respective bandwidth guarantees should be modeled, a logical level view is also provided.

The expressiveness of the model is defined by the power of the modeling language. Therefore, the goal is to use a language that is capable of expressing as much as possible about the system, and at the same time keep the depiction simple. In general we can differentiate between static and behavioural modeling. In the first case we can model elements like power consumption, space occupation, ROM memory as well as interconnections between systems or failure probabilities, only to name a few possibilities. But when describing of the dynamics of a system is necessary, behavioural modeling is important. It gives the possibility to model e.g. state machines, communication flows and execution times. Depending on the possibilities of the modeling language various analysis are possible. E.g. if the possible power consumption of a mobile handset and its subcomponents is known, as well as the capacity of the rechargeable battery, which is itself constrained by the form factor of the handset, the runtime of the device can be anticipated. This example sounds like a simple problem statement, but when the different states (standby, active data connection, different networks) of the handset are incorporated which will result in different power demands, it becomes clear that a good model will help to capture all eventualities.

Another possible feature of a modeling language is automatic code or code skeleton generation, which would enable faster product development and mean less developer-caused bugs. This feature is mainly dependent on the specificity of the modeling language. Like in SysML the language is kept very general and is not capable of describing very specific behaviour that would enable code generation. In contrast, SCADE is specifically designed for automatic code generation, but on the other hand lacks general purpose features.

Another important point is the possibility to include and trace requirements in the model, since they describe the basic system constraints. Especially in projects with lots of people working on, and various shareholders included, this feature is a must. The immediate connection between pre-defined requirements and the model of the system decreases communication overhead, possible requirement-to-model transition failures and gives the opportunity to backtrace problems, shortcomings and inadequacies from the occurring place to the corresponding higher level requirement. The strict handling of requirements is especially important in the safety critical domain.

In addition, a modeling language is only as strong as the supporting tools. A syntax for the model can be defined very fast, but the tools needed for utilizing all the possible features of the modeling language to support a fast and clean modeling are sometimes hard to find. This is especially the case for features like automatic code generation or timing checks. Therefore the choice of the modeling language is not only defined by the expressiveness of the language, but also by the availability of proper tools, which is primarily defined by the industry and academic usage of the modeling language.

Since this is a thesis about modeling a system which uses Time Triggered Ethernet as communication platform, it has to be decided what features of TTEthernet have to be modeled, and which can be used in an abstracted form in the model.

## 4.2 Specific Issues

One of the main properties of TTEthernet is the use of the time triggered paradigm. It gives the chance to abstract data communication to the level of message state synchronization with temporal guarantees. E.g. the feature of guaranteed correct message order delivery comes in handy. Hereby this property can be presumed and the possibility of certain failures doesn't have to be taken into account.

In general almost all modeling languages have a feature for message sending. The behavioural description is mostly based either on a asynchronous event triggered approach or on a strict synchronous data flow approach as seen in SCADE/Lustre. TTEthernet has the possibility to send either way, event triggered, time triggered or with bandwidth guarantees. So with respect to message passing the following items would be necessary:

- Describing the physical and logical topology of the network,
- assigning the different traffic classes to channels,
- describe the different channel properties like cycle intervals and bandwidth guarantees,
- assign the channels to actual hardware nodes, switches and physical links, including redundancy.

After a sufficient and proper description of the network, some analysis would be required:

- Checking the feasibility of the network with respect to bandwidth consumption and timing demands of the network based on the given parameters,
- calculating end-to-end send delays and maximum possible jitter for every channel based on the hardware layout and check if they conform with the requirements,
- executing a fault analysis according to the fault hypothesis to validate fault-tolerant mechanisms.

If the model is validated with respect to the defined requirements, there are several more possibilities:

- Automatic code skeleton generation, including the network interface initialized with proper values to satisfy the model,
- automatic configuration of the network infrastructure (i.e. switches and endsystems),
- automatic configuration of a partitioned OS on every node to correspond to the sending/receiving interval,
- automatic generation of test cases,
- automatic generation of documentation and network parameters to support testing and debugging.

These points are probably not covered by any standard modeling language. It is merely a task of domain specific and even application specific language extensions and tools. E.g. the automated code skeleton generation depends on the language used and has to be implemented differently for every one. But this complex theme will not be topic of this thesis.

## 4.3 The Choice

In this thesis two different modeling languages will be chosen to implement a simple TTEthernet network consisting of three endsystems and one switch. Although TTEthernet networks can consist of hundreds of endsystems and switches – possibly in a redundant configuration – we will stick with a smaller layout for the sake of simplicity. The choice was based on multiple factors, such as:

- Possibility to model different network communication paradigms. In our case all three message classes want to be modeled.
- Possibility to add properties to communication links. Such as bandwidth consumption, delays, message typing.
- Possibility to describe the physical connections, as well as the logical interconnections to get a proper abstraction level.
- Possibility to model as many TTEthernet properties as possible with the highest possible abstraction to keep the model simple and clear.

For the tool support the following features are wanted:

- Model consistency checking.
- As many built in property checks as possible (e.g. checking if the power consumption requirement is met)
- Easy extension of property checks (e.g. checking if the bandwidth of all channels on one physical link does not exceed the overall budget)
- It should be well tested and used in the industry and/or academics, with an anticipated growing acceptance rate in these fields.
- Open Source is highly favored and leaves the possibility for tailor made extensions.
- Features that support team collaboration for large projects.

It is clear that at the current stage of development, no modeling language and its supporting tools will satisfy all these requirements. Which points a mandatory and which points are obligatory are primarily defined by the nature of the project. E.g. in small projects with little requirements, property checking like overall power consumption could be done manually. Whereas in larger projects, with high complexity and many subcomponents, this feature could become highly important, so that costs of implementation of certain checks would be accepted.

In this thesis the choice was made to give a TTEthernet demo implementation in the SAE Architecture Analysis and Design Language (AADL) as high-level modeling language and the Safety

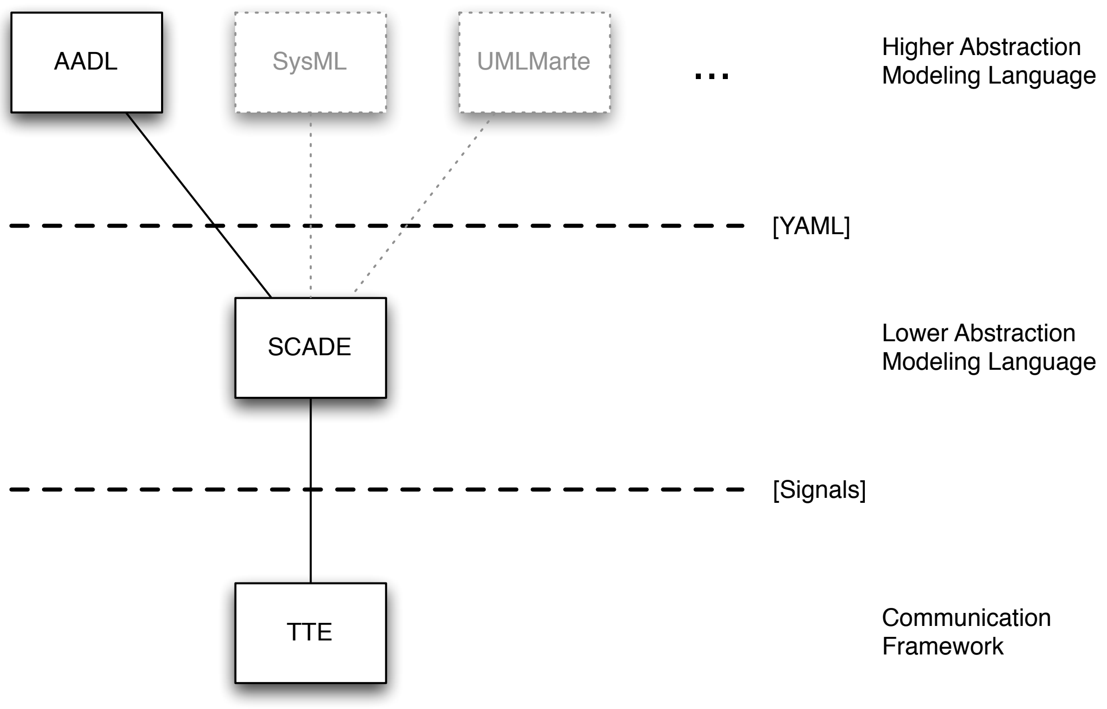

AADL
SysML
UMLMarte
...
Higher Abstraction Modeling Language
[YAML]
SCADE
Lower Abstraction Modeling Language
[Signals]
TTE
Communication Framework

**Figure 4.1:** modeling language abstraction levels

Critical Application Development Environment (SCADE) as low-level modeling language. Figure 4.1 shows these two languages in conjunction with TTEthernet and their interfaces. Note that the interface between high- and low-level modeling language is specified as YAML [27] file. For reasons of better interoperability to other high-level modeling languages. The interface to TTEthernet is based on the Signal notion from SCADE, since this is the only necessary information exchange.

# Chapter 5

# AADL

AADL was chosen because of the following reasons:

- The language is standardized by the Society of Automotive Engineers and specifically designed for the use in the safety critical and embedded domain. Thus it has features specifically designed for interconnection networks and different message passing paradigms, making it a natural candidate for designing TTEthernet applications. It is explicitly allowed and encouraged to add timing information to the model (e.g. interval definition of periodic tasks, message propagation delays, etc.)
- An open source tool named the "Open Source AADL Tool Environment" (short: OSATE) is available, which is based on the Eclipse framework and supports graphical modeling, automatic model consistency checking and various general purpose property checking. An integration in TOPCASED (The Open-Source Toolkit for Critical Systems) is possible and comes in very handy when dealing with larger projects. It should be noted that support for AADL in version 2 is not given yet, but is in development.
- OSATE supports a plugin mechanism for adding custom model analysis.
- It is possible to model hardware components, as well as logical elements and assign them to each other.
- Its networking elements and connections are defined in the behavioural domain, featuring different message passing patterns including their behaviour in the time domain.
- The modeling follows an object oriented paradigm and building packages is possible.
- Own property sets can be defined easily, allowing their static checking for model analysis.
- The language is designed to support large projects with many subcontractors on various tiers as well as stakeholders.

One thing important to keep in mind is the fact that AADL is a rather young language, without many reference projects and it's supporting tools are still not as powerful as tools of other modeling languages. But the industry and academics have shown their support for this language in recent years [28]: Boeing, Airbus, European Space Agency, Ford, Toyota, Rockwell-Collins, University of Pennsylvania, Carnegie Mellon, etc.. Many of these companies and universities also announced that they will strongly enforce the usage of AADL in future projects.

The risk inherent in a modeling language, which is not well tested in many hundreds or thousands of projects, can not be neglected. But nevertheless, a language that is specifically designed for safety critical embedded systems without the need of workarounds sounds very promising. It has to be noted that AADL was primarily designed for aerospace applications and formerly known as *Avionics Architecture Description Language*. Hereafter it will be tried to map features of TTEthernet to constructs provided by AADL.

All properties defined in the following sections will be included in an own property set where all TTEthernet properties will be defined and unified. This is stated, because certain standard properties may be part of the standard AADL property set already. This choice was made in favour of consistency and clarity. The range description of any property set is just a suggestion and may change depending on the network configuration and use case.

Further, it has to be noted that the diagram representation of systems and subsystems is only displayed for easier reading and better overview. Only the code is consistent and describes the correct and full implementation. Reason for that is the buggy implementation of the graphical editor and AADL parser that comes with OSATE. On the one hand it interprets not all descriptions (like extensions and implementations) and on the other hand it does not support the modeling of more advanced features. Excerpts of the code will be shown where useful, a complete model will be printed in Chapter **??**.

## 5.1 Implementation

### Physical Interconnections

One main feature of TTEthernet is the paradigm of different traffic classes with their respective features. In AADL, networks can be described as physical entities, providing network access to various other physical objects like systems or devices. On this level, a classification of various network classes does not make sense. Only properties connected to the physical target will be defined:

| | |
|---|---|
| *Bandwidth* | { 100Mbit \| 1Gbit } |
| *Length* | 0.1m .. 1000m |
| *Mediatype* | { 100BaseT \| 100BaseFX \| 100BaseSX \| 1000BaseTX \| 1000BaseSX \| 1000BaseLX } |

**Figure 5.1:** Properties for TTE Bus

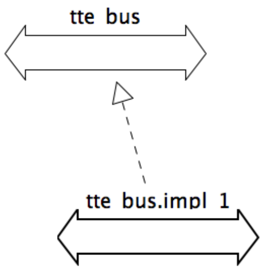


**Figure 5.2:** AADL bus definition

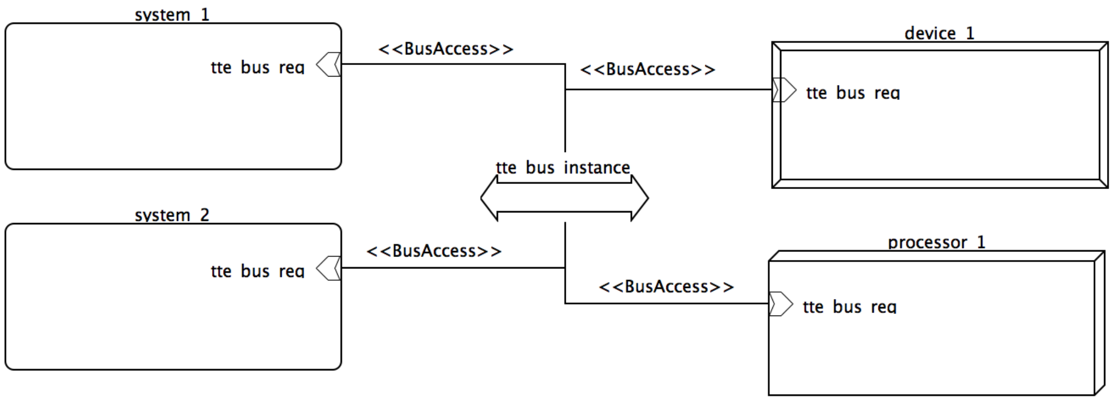


**Figure 5.3:** AADL entities requiring bus access

Figure 5.2 shows a *bus* definition, and 5.3 an example of various entities connected via a bus. It should be kept in mind that the showed systems, device and processor are instances of implementations (not shown). The definition requires a bus access, and thus all instances of implementations do as well.

## Channels and Traffic Classes

In TTEthernet, the concept of channels is important. It gives the highest tier of abstraction when thinking about message passing on a logical level. In AADL, message passing is done via *ports*, which can be defined as follows [29]:

- *Data Port*: Used for state based data transmission among entities without queueing.
- *Event Port*: Used for event communication which have queue semantics. E.g. dispatches of aperiodic threads. No additional data is transmitted
- *Event Data Port*: Combined semantics. Message passing with queueing; Every data is associated with an event.

| | |
|---|---|
| *Traffic Class* | { TT \| RC \| BE } |
| *Interval (if TT or RC)* | 100us .. 10s |
| *Message Size* | 46byte .. 1500byte |
| *Data Type* | { int_16 \| int_32 \| int_64 \| real_64 \| boolean \| blob \| enum } |
| *Instances* | 1 .. 3 |

**Figure 5.4:** Properties for TTE port

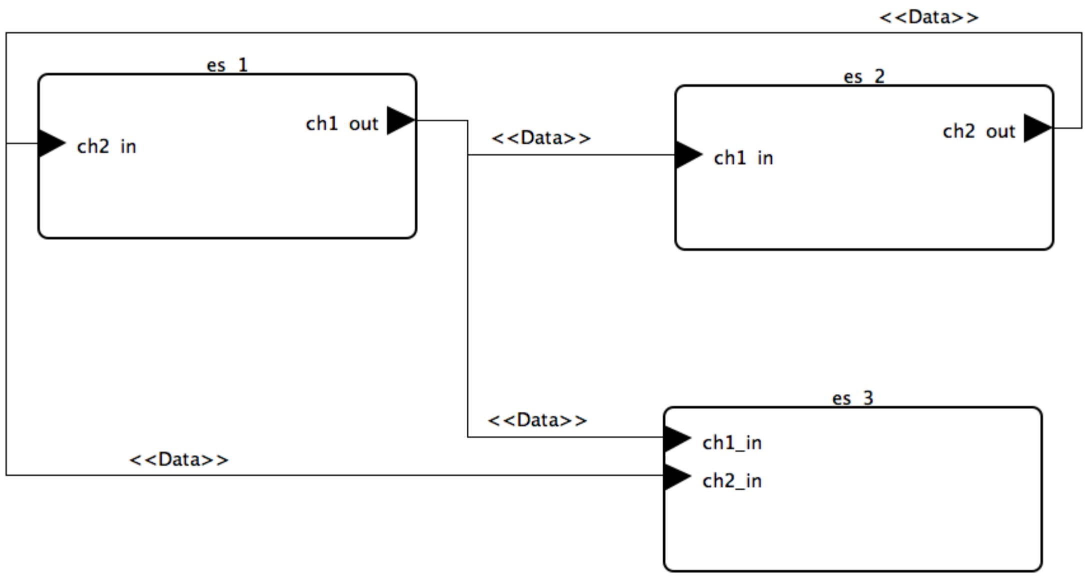


**Figure 5.5:** TTEthernet endsystems connected via data ports

*Data Port* semantics are the most suitable representation of the Time-Triggered traffic class, since we do not have queueing and can connect every port with a periodicity that describes the sending interval. Whether Best-Effort and Rate-Constrained traffic can be seen as *Data Port* traffic or *Event Data Port* with regards to the queueing semantic is a difficult question. Since Rate-Constrained traffic gives guarantees about the bandwidth, but not about the receiving points in time, it is possible that some messages can be queued in the switch (i.e. the Time-Triggered traffic blocks Rate-Constrained communication temporally). This would be equal to queueing at the receivers port. Theoretically, no Rate-Constrained traffic should be dropped under any circumstances, but due to the difficulty to find the maximum queue size for every switch for every communication scenario and then to provide this queue size in hardware, it cannot be guaranteed. This leaves the question, which port implementation would be suited better. But since a property in the *tte property set* that defines the property of the traffic class is provided, at least property checking can be done.

Table 5.4 gives a list of the properties associated with the AADL ports. The property *Instances* gives the grade of redundancy. 1 stands for no redundancy of the channel, 2 for dual, 3 for triple

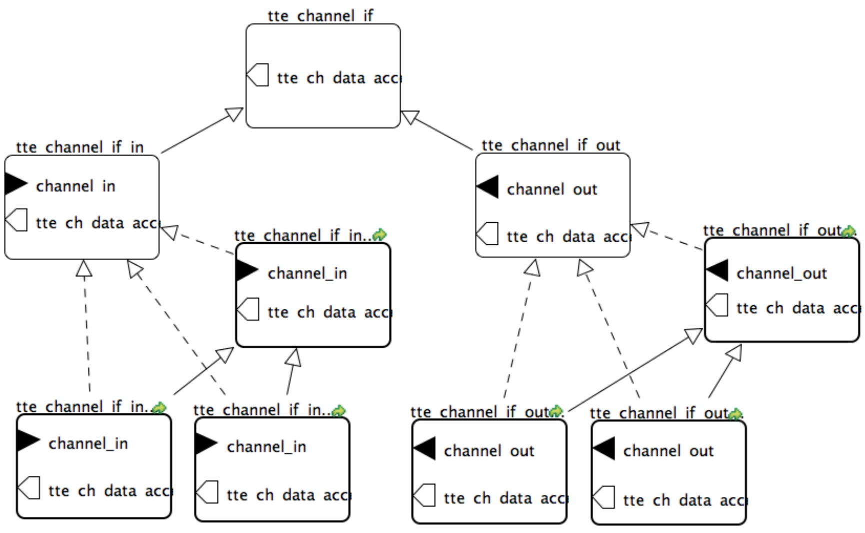


**Figure 5.6:** TTEthernet channel interfaces

redundancy. The reason why redundancy is a channel property and not a hardware bus property is that with TTEthernet it is possible to implement single channels redundantly. Otherwise it would be difficult to define which ports or channels on an end system are sent over multiple buses and which are not. The other properties should be self-explanatory.

Figure 5.5 shows a simple TTEthernet system, which shows the data flow of every channel. It has to be noted that every channel only has one sender and has at least one receiver.

With the concept of channels and ports, the inheritance features of AADL become practical. A channel port, can be divided into incoming and outgoing channel ports. They can then be further categorized in the different channels that are present on the system, inheriting the feature of the incoming or outgoing port of the higher level. This feature can also be used to create channel classes; e.g. channel class A for video streams with the same bitrate, and channel class B for sensor data with the same bitrate and presentation; channels 1 and 2 can then be derived from class A, and channels 3 to 13 from class B. Especially in systems with a high number of nodes, this feature is not only convenient but necessary.

Figure 5.6 shows the inheritance of a channel interface. What every interface has in common is the provided access to a data object, denoted by the top object. Further, the interfaces get divided into in- or outbound channel ports. The internals of such an in- or outbound port are described in the implementation of an interface. The separate channels are then implemented on the basis of the in- or out-channel definition. The implementation, which carries the internal subcomponents, is shown in figure 5.7: A channel interface contains a data component and a process component where the first is responsible for storing the channel data and the second is responsible for dispatching or receiving the data. The properties described in Table 5.4 are connected to the implementations of the channels.

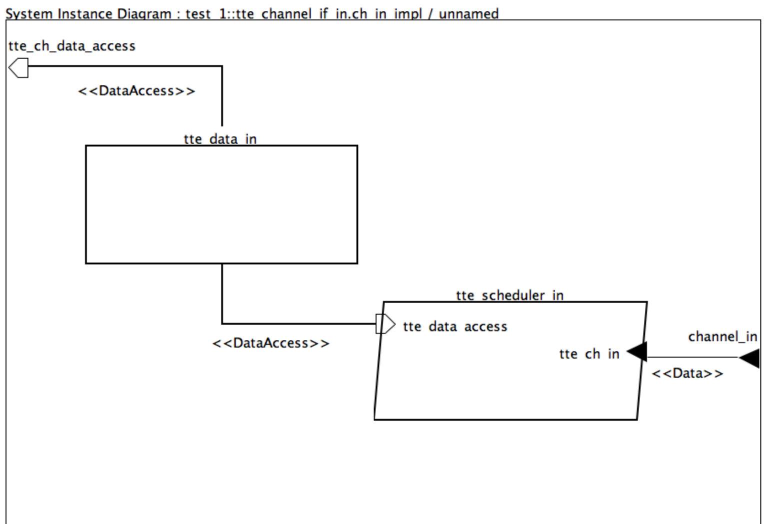


**Figure 5.7:** TTEthernet channel interface subcomponents

## 5.2 Further thoughts

As already mentioned, AADL is suitable for large projects with many stakeholders. In a project where TTEthernet is used as communication network it would be advantageous if the corresponding AADL model could simply be added as package where needed. The needed property settings could then be defined by the system designer.

Automatic schedule generation for the TTEthernet network from the AADL model should also be an important goal, since this would reduce the design effort significantly. Additionally, the schedulability and other checks connected to the generation of a valid schedule could already be done within AADL in the early design phase.

# Chapter 6

# SCADE

In contrast to AADL, which focuses on a static and overall model description based on entities and properties, SCADE is a language merely used for behavioural description. Well known and used in the safety critical domain, it is obvious that it would be used in projects where TTEthernet is used as interconnection network. Different to AADL is the way TTEthernet embeds with SCADE. Where in the former, TTEthernet is a element of the model, which is also by itself described by the modeling language including all necessary properties, in the latter TTEthernet is an external model that has to satisfy properties given from the SCADE model. To be precise, signals in the SCADE model should be transferred from one node to another. Message size and send frequency are given by the SCADE model, respectively the runtime system, and have to be satisfied by the network model.

The problem hereby is the interface between these two models – in the design phase, as well as in the execution phase. Firstly, the SCADE suite has to be provided with the necessary variable names that should be used as external interfaces in order to connect to the TTEthernet interface. Secondly, a middle layer has to be provided, which packs the signals in the corresponding TTEthernet frames.

Figure 6.1 shows a proposed workflow for designing a project in SCADE, while communicating with TTEthernet. The Databases (or Tables) to the left describe two different relationships. The *Signal DB* (see section 8) describes which signals exist and to which *VL ID* they should be assigned, and the *TTE-Schedule* describes which *VL IDs* exist and what their dataflow is. The *TTE-Scade Integrator* is responsible for converting it to a format which can then be read by SCADE.

The other conversions affect the *Gluecode*, which is responsible for Signal packing and interacting with the TTEthernet driver. This layer is build to be statically configured. Meaning, that the configuration of the signal packing has to be defined at compile time. An alternative “lookup solution”, where the *Gluecode* would be configured at runtime, would introduce unnecessary configuration complexity and runtime load. This is of high importance, as that this code is frequently executed and could possibly impose a very high resource utilization, especially in use cases with high sending/receiving rates but small packets.

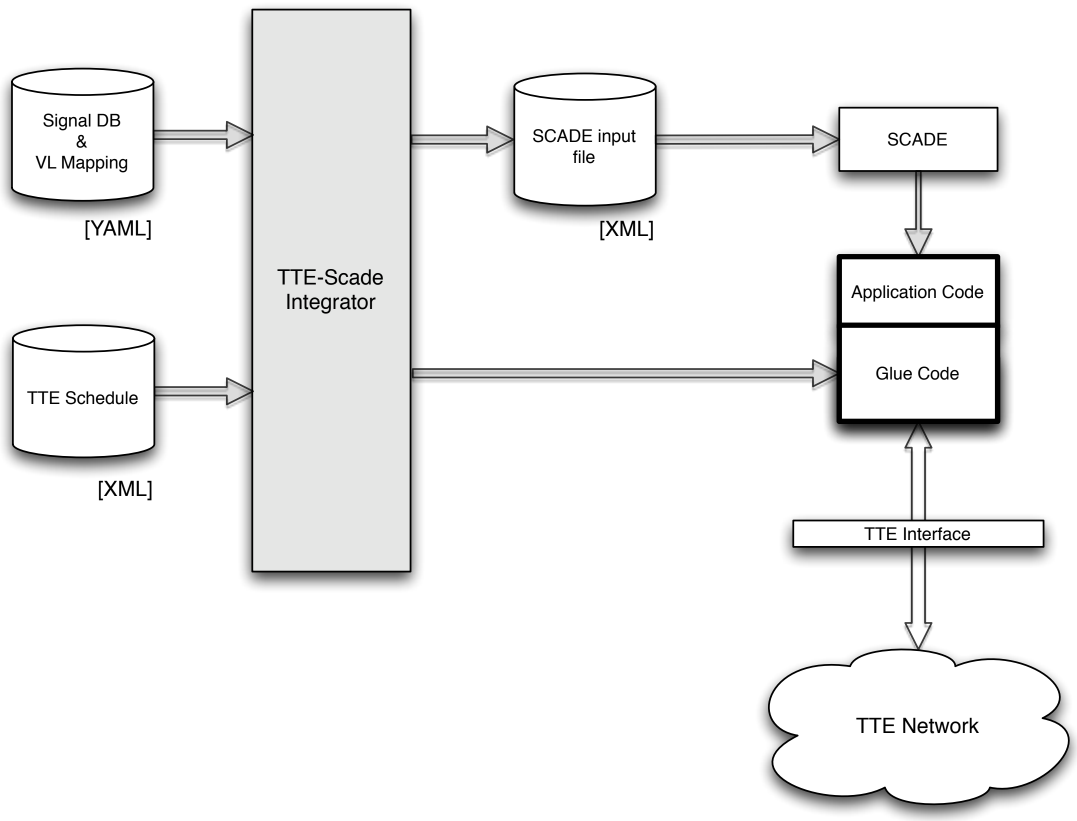


**Figure 6.1:** SCADE TTEthernet workflow and communication

## 6.1 Implementation

### Signal DB to SCADE

The first task is to provide a SCADE interface in the design phase. In our case, the *Signal DB* is described in the YAML format [27] and contains the following information for every signal (see listing 8.1):

- *S_ID*: Signal ID. Unique Signal identifier name.
- *S_TYPE*: Signal Type. Necessary for the SCADE import.
- *S_DIR*: Signal Direction. Necessary for the SCADE import.
- *VL_ID*: Virtual Link id the signal should be packed in.
- *size*: Size in bits. Necessary for the *Com-Layer*.

- *offset*: Offset of the Signal in the VL Frame. Necessary for the *Com-Layer*.

With items *S_ID*, *S_TYPE*, *S_DIR* there is enough information to generate a SCADE template which then can be used for application coding. Important is, that the signal names responsible for in- and output do not get changed in SCADE, otherwise the *Gluecode* will fail in binding the correct signals. Since SCADE does not have a feature for importing input or output signals, a workaround has to be used. SCADE stores its model data in *.xscade* files, which are in the XML format. This file can easily be generated including the necessary in- and outputs. A drawback is though, that once the file is used by SCADE and saved again, no further change concerning the input and output signals can be made. This is due to the fact that as soon as this file gets saved by SCADE, internal unique identifiers will be added and rearranging of the signals happens. Figure 8.3 shows a generated *.xscade* file.

The items *VL_ID*, *size* and *offset* define the packaging information needed to send the signals accross the network. The size defines the actual size of the type, meaning that if 16bit are given for an *integer* signal, only 16bit will be sent, regardless if an *integer* is possibly 32bit wide on the target machine. The system programmer has to take care of that fact.

Figure 6.2 shows a screenshot of SCADE after importing the generated *.xscade* file, containing *upper_floor_node_1* and its interfaces originated from the *Signal DB*. A recommended mode of operation would be to use as few child-operators as possible for the top layer operator (i.e. *upper_floor_node_1*), since a possible re-generation of the top operator would mean that all child-operators have to be connected again.

### Signal DB to TTEthernet Interface

The *Gluecode* is the connection between the TTEthernet driver and the SCADE interface. For this prototype implementation it was chosen to use a simple code generator, which outputs a simple C File that encapsulates the SCADE function and can then be cross compiled and run on the embedded hardware. The basic architecture of this generated file looks as follows:

- *Allocation*: The Frame Buffers and the SCADE signal structs are statically allocated according to their space demands.
- *Receiving*: Incoming Frames are received from the TTEthernet interface and stored in the frame buffer.
- *Unpacking*: Frame Buffers get dissected and stored in the SCADE structs.
- *SCADE call*: The main SCADE function gets called and the received signals get provided. After the call, the signals that have to be sent are stored in the output struct.
- *Packing*: The computed signals get packed together and stored in the outgoing frame buffers.

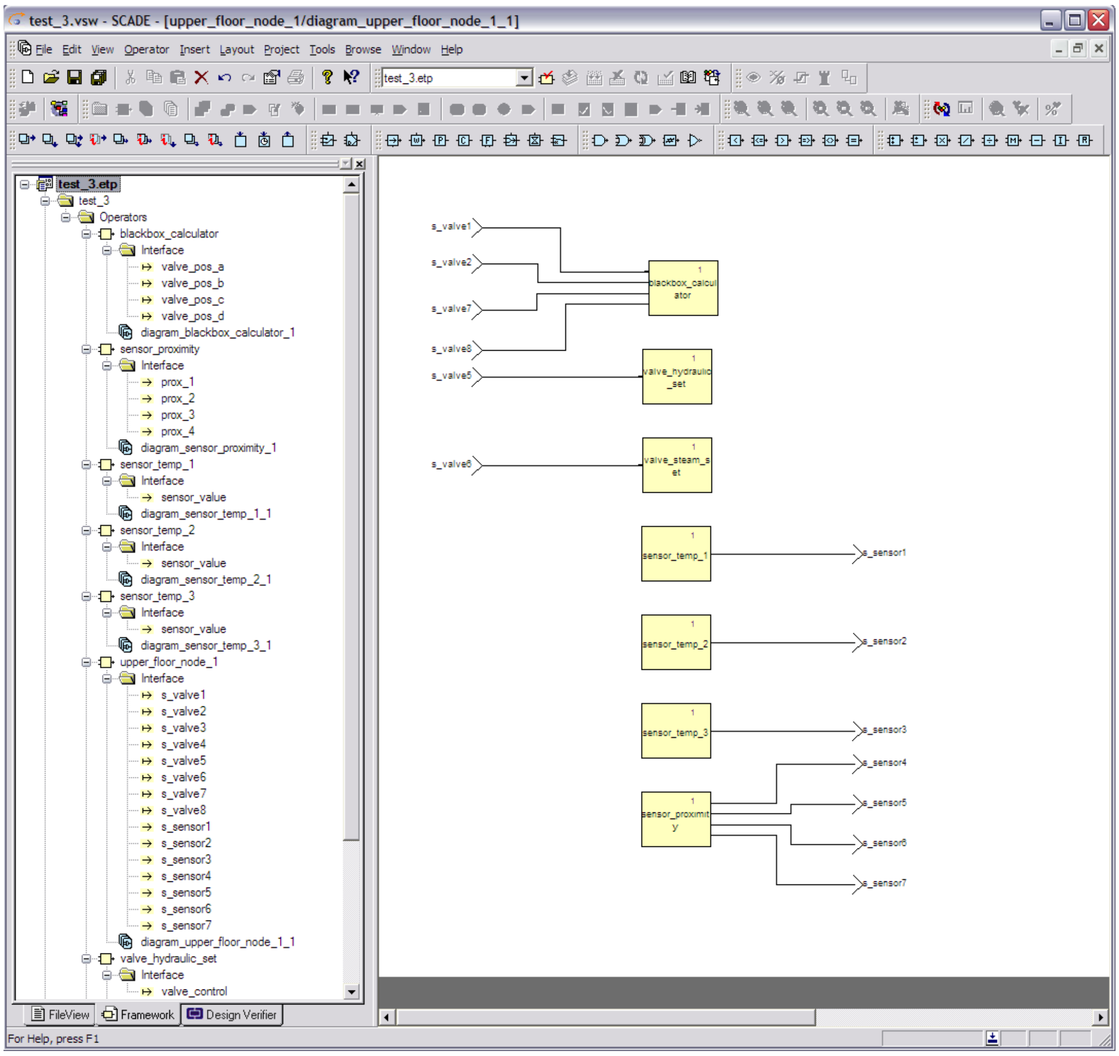


**Figure 6.2:** SCADE example screenshot

- *Sending*: The TTEthernet interface gets called to send the frame buffers via the corresponding Ports/VL_IDs.

Listing 8.4 shows the generated C file for the already presented YAML file and corresponding network description. Particular attention should be paid to the packing of boolean signals. Since in SCADE, boolean variables are simply represented by an *integer*, this would impose an significant communication overhead. Therefore the “packaging” of these data type is necessary. It should be noted that in this implementation different endianess on different nodes in the network is not supported. Figure 6.3 shows an exampling for packing multiple signals in one Frame. Coloured bits indicate a usage of them.

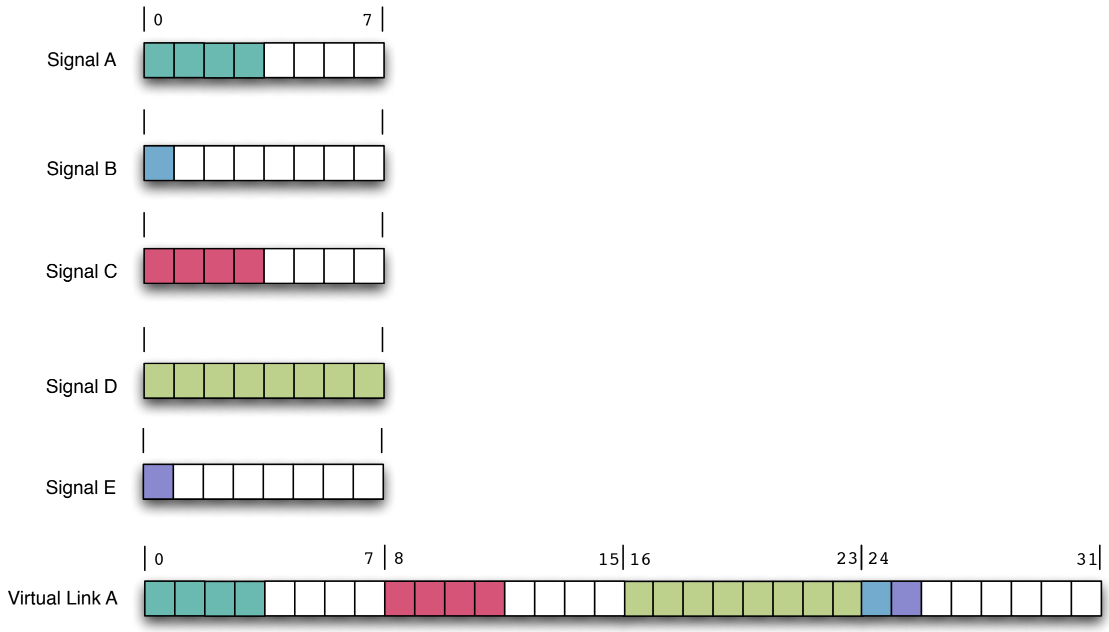


**Figure 6.3:** Sample packaging of the *gluecode*

This generated function is then called by the runtime system – which can be a operation system with cyclic scheduling or even a partitioned OS. Keep in mind that the calling frequency gets defined by the scheduler and not by the model or code itself. Additional parameters from the OS can be passed to the function, although this function is not used yet.

# Chapter 7

# Conclusion and further work

At first different modeling languages were examined with respect to their capabilities, use cases and distribution. It was discovered that every modeling language is covering a different abstraction layer and different usages. Their usefulness in covering the specific demands of real time systems was also shown. Further it was discussed that there is no single modeling language that covers all the different abstraction layers at once sufficiently in respect to their accurateness and use case coverage.

In Chapter 3 an overview of Time Triggered Ethernet was given and some unique features including the different traffic classes were explained, and furthermore it's handling of time, network management communication, send order re-establishment and possible topologies. It was showed why TTEthernet is inherently suited for embedded real-time solutions and examples were included.

Further, special demands of TTEthernet with respect to modeling were identified, including the particular demands deriving from its use cases and its area of application. Consequential two different modeling languages were chosen that are the most promising and that combined give the possibility to model from a high abstraction down to a low-level behavioural description. Not only the formal possibilities of the modeling languages, but also the extend of acceptance in the scientific and industrial community were a decision guidance.

With the first choice, the SAE Architecture Analysis and Design Language (short AADL), a simple system from an high abstraction level, including the physical as well as the logical view was modeled. The TTEthernet architecture on these two views was matched to appropriate AADL entities including a suitable property description. Further, it was shown that when a certain amount on tool programming and network property description is done, extensive feasibility checks can be achieved.

For the Safety Critical Application Development Environment (short SCADE), a workflow and a tool for automatic configuration was presented. Starting with a list of signals, a SCADE skeleton was produced where the application code can then be implemented. Given this envelope of inputs and outputs, a glue code was produced that couples signals to TTEthernet frames which then can be called by the operating system in the predefined cycle time.

The combination of AADL and SCADE as modeling languages gives a possible and feasible way to implement a system in the embedded real-time domain. Although AADL still gives space for further improvement on the tool side, its development is promising. Additionally, on this higher abstraction layer there will always be the need for adaptions and extensions to support certain use cases and projects. In contrast, the use of SCADE on the lower abstraction level in conjunction with the here presented toolset gives a already usable solution.

Summarizing the advantages of this approach: The use of modeling languages in the domain of real-time embedded systems is advantageous and sometimes even obligatory. Especially in projects with many stakeholders and developers the benefit of using a higher abstraction modeling language can be measured in a shorter design phase, more flexibility and overall fewer problems during development. On the lower abstraction level the advantages of modeling languages express in fewer coding errors and faster development. The use of Time Triggered Ethernet brings a further abstraction on the level of periodic message exchange which fits optimally with the signal paradigm in SCADE.

## 7.1 Shortcomings

In theory, all modeling languages and their workflows sound highly promising and seem to boost development times and code quality. But there are several serious shortcomings that are less derived from a specific modeling language, as from the paradigm itself.

In general, modeling languages try to serve all (necessary) possibilities of use cases and system properties and map them to language intrinsic constructs. This may be sufficient for a very large group of projects, especially for desktop software – thus the heavy use of UML in this area – but projects in the embedded domain do have different needs:

- The hardware often is not fixed, and has to be developed parallel to the software.
- The use of different programming languages – and also hardware description languages – is highly probable.
- Configuration of hard- and software before operation is often necessary.
- Then software has to be written in order to support the configuration.
- The runtime environment often has crucial impact on the application itself, thus the application can not be seen as an isolated entity.

This is just an excerpt of what the modeling language would have to cover for certain projects or use-cases. In general, it can be said that it is almost impossible to cover all necessary information and properties with one language. In practice, several modeling languages will be used, from formal languages down to natural language descriptions. Besides, the effort of customization to support certain demands sometimes outruns the benefits when using a modeling language.

## 7.2 Further work

Not covered by this thesis is the automatic generation of the *Signal DB* from the AADL model. This would further increase the extent of integration of the higher-level modeling language. But it has to be noted that an independent layer like YAML is still strongly recommended.

Current development on the level of higher abstraction modeling languages also include SCADE System by Esterel, which is available in Version 1. It is based on SysML and covers also the description of non-software entities. Further, it could also be used as a source for the network description.

Further work should also consider the use of Modelbus [30] which addresses the issue of interaction of different modeling languages. Although it is focused more on "large scale" software rather than embedded systems, it should be kept in mind. Especially the integration in the *Eclipse Modeling Framework* is highly interesting, since more and more tools for modeling languages are already or are going to switch to this platform.

# Chapter 8

# Listings

## 8.1 TTE-SCADE Integrator Code

The source code of TTE-SCADE Integrator is supplied separately, and can also be obtained anytime by asking the author of this thesis.

## 8.2 Signal Database

```
# YAML test file for SCADE-Link
---
document_info:
    mapping:          m_example_1
    nodename:         upper_floor_node_1
    document_version: 1
    content_version:  1
    contact:          Valentin Ecker

signals:
  - s_id:   s_valve1         # signal name
    s_type: int              # scade type
    s_dir:  in               # signal direction [in|out]
    vl_id:  vl_valves_cat_1  # virtual link name, corresponds to TTE_Tools
    size:   8                # in bits
    offset: 0                # offset in VL

  - s_id:   s_valve2
    s_type: int
    s_dir:  in
    vl_id:  vl_valves_cat_1
    size:   16
    offset: 8

  - s_id:   s_valve3
```

```
    s_type: int
    s_dir:  in
    vl_id:  vl_valves_cat_2
    size:   16
    offset: 0

  - s_id:   s_sensor1
    s_type: real
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   16
    offset: 0

  - s_id:   s_valve4
    s_type: int
    s_dir:  in
    vl_id:  vl_valves_cat_3
    size:   32
    offset: 0

  - s_id:   s_valve5
    s_type: bool
    s_dir:  in
    vl_id:  vl_valves_cat_3
    size:   1
    offset: 32

  - s_id:   s_valve6
    s_type: bool
    s_dir:  in
    vl_id:  vl_valves_cat_3
    size:   1
    offset: 33

  - s_id:   s_valve7
    s_type: bool
    s_dir:  in
    vl_id:  vl_valves_cat_3
    size:   1
    offset: 34

  - s_id:   s_valve8
    s_type: bool
    s_dir:  in
    vl_id:  vl_valves_cat_3
    size:   1
    offset: 35

  - s_id:   s_sensor2
    s_type: real
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   16
    offset: 16

  - s_id:   s_sensor3
    s_type: real
    s_dir:  out
    vl_id:  vl_sensors_cat_1
    size:   32
    offset: 0

  - s_id:   s_sensor4
    s_type: bool
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   1
    offset: 16

  - s_id:   s_sensor5
    s_type: bool
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   1
    offset: 17

  - s_id:   s_sensor6
    s_type: bool
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   1
    offset: 18

  - s_id:   s_sensor7
    s_type: bool
    s_dir:  out
    vl_id:  vl_sensors_cat_2
    size:   1
    offset: 19
...
```

**Listing 8.1:** Signal DB [YAML]

## 8.3 Network Description

```
<?xml version="1.0" encoding="UTF-8"?>
<ns5:NetworkDescription
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:buf="http://www.tttech.com/Schema/TTEthernet/Network_Description/
    Buffering/3.1/201204171613"
  xmlns:c="http://www.tttech.com/Schema/TTEthernet/Network_Description/
    Constraint/3.1/201204171613"
  xmlns:nd="http://www.tttech.com/Schema/TTEthernet/Network_Description_V1"
```

```
xmlns:topo="http://www.tttech.com/Schema/TTEthernet/Network_Description/
  Topology/3.1/201204171613"
xmlns:vl="http://www.tttech.com/Schema/TTEthernet/Network_Description/
  Virtual_Links/3.1/201204171613"
xmlns:ns0="http://www.tttech.com/Schema/TTEthernet/
  Network_Description_V1_V20110811_01"
xmlns:ns1="http://www.tttech.com/Schema/TTEthernet/
  Network_Description_V1_V20110817_01"
xmlns:ns2="http://www.tttech.com/Schema/TTEthernet/
  Network_Description_V1_V20110826_01"
xmlns:ns3="http://www.tttech.com/Schema/TTEthernet/
  Network_Description_V1_V20110829_01"
xmlns:ns4="http://www.tttech.com/Schema/TTEthernet/Network_Description_V2"
xmlns:ns5="http://www.tttech.com/Schema/TTEthernet/Network_Description
  /3.1/201204171613"
name="scade_integrator_example_network"
interfaceVersionNumber="1"
transmissionSpeed="1000Mbps"
redundancy="2"
ctMarker="ab:ad:ba:be"
enableDynamicRouting="false"
ctMask="ff:ff:ff:ff"
createUnknownDefaultRoutes="false"
xsi:schemaLocation="http://www.tttech.com/Schema/TTEthernet/
  Network_Description/Buffering/3.1/201204171613 ../
  html_schema_doc_and_schemas/ND/ND_Buffering.xsd http://www.tttech.com/
  Schema/TTEthernet/Network_Description/Constraint/3.1/201204171613 ../
  html_schema_doc_and_schemas/ND/ND_Constraint.xsd http://www.tttech.com/
  Schema/TTEthernet/Network_Description/Topology/3.1/201204171613 ../
  html_schema_doc_and_schemas/ND/ND_Topology.xsd http://www.tttech.com/
  Schema/TTEthernet/Network_Description/Virtual_Links/3.1/201204171613 ../
  html_schema_doc_and_schemas/ND/ND_Virtual_Links.xsd">
<metaData
  dataid="TTE-Scade_Integration_v1"
  description="Example TTE Network for TTE-Scade Integration"
  date="2012-05-03T00:00:00.000+0200"
  author="Valentin Ecker"
  version="1"
  company="UT Vienna"
  copyrightText=""/>
<processingInstructions>
  <schedOption
    Key="seed"
    Value="12345"/>
  <schedOption
    Key="minimum_delta_r"
    Value="2"/>
</processingInstructions>
<syncDomain
  name="syncDomain_1"
  refClusterPeriod="#//@period[name='STATISTIC_PERIOD']"
  integrationCycleDuration="1000000ns"
  faultTolerance="0FTSI_2SM"
```

```
    precision="5008ns"
    fullCBG="true"
    value="0">
    <syncPriority
      name="syncPriority_1"
      value="1"/>
  </syncDomain>
  <device
    xsi:type="topo:Switch"
    name="sw0"
    syncRole="syncCompressionMaster"
    refSyncPriority="#//@syncDomain/@syncPriority[name='syncPriority_1']"
    deviceTarget="TTE_Dev_Switch_12port_1G">
    <port
      name="sw0_P1"
      type="P1"/>
    <port
      name="sw0_P2"
      type="P2"/>
    <port
      name="sw0_P3"
      type="P3"/>
    <port
      name="sw0_P4"
      type="P4"/>
    <port
      name="sw0_P5"
      type="P5"/>
    <port
      name="sw0_P6"
      type="P6"/>
    <port
      name="sw0_P7"
      type="P7"/>
    <port
      name="sw0_P8"
      type="P8"/>
    <port
      name="sw0_P9"
      type="P9"/>
    <port
      name="sw0_P10"
      type="P10"/>
    <port
      name="sw0_P11"
      type="P11"/>
    <port
      name="sw0_P12"
      type="P12"/>
    <port
      name="sw0_PMGMT"
      type="PMGMT"/>
    <port
```

```
      name="sw0_PSYNC"
      type="PSYNC"/>
    <managementInterface
      sourceAddress="02:02:02:02:04:2F"
      unlockDestAddress="02:02:02:02:08:2F"
      channel="1">
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:21"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:2F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:22"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:2F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:23"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:2F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:24"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:2F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
    </managementInterface>
    <managementInterface
      sourceAddress="02:02:02:02:04:4F"
      unlockDestAddress="02:02:02:02:08:4F"
      channel="2">
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:41"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:42"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:43"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:44"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
    </managementInterface>
  </device>
  <device
    xsi:type="topo:EndSystem"
    name="upper_floor_node_1"
    syncRole="syncMaster"
    refSyncPriority="#//@syncDomain/@syncPriority[name='syncPriority_1']"
    deviceTarget="TTE_PMC_ESys_1G">
    <port
      name="upper_floor_node_1_P1"
      type="P1"/>
    <port
      name="server_PMGMT"
      type="PMGMT"/>
    <port
      name="server_PSYNC"
      type="PSYNC"/>
    <port
      name="server_PHOST"
      type="PHOST">
      <macInterface
        name="MAC_INF_0_for_server"
        address="01:02:03:04:05:06"/>
    </port>
    <managementInterface
      sourceAddress="00:00:00:00:00:00"
      unlockDestAddress="00:00:00:00:00:00">
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:41"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:42"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
      <macAcceptanceEntry
        acceptanceMacAddress="02:02:02:02:00:43"
        addressType="nonCriticalTraffic"
        unlockEnabled="true"
        resetEnabled="true"
        responseDestMacAddress="02:02:02:02:08:4F">
        <accessControl
          page="0"
          writeEnable="true"/>
        <accessControl
          page="1"
          writeEnable="true"/>
        <accessControl
          page="2"
          writeEnable="true"/>
        <accessControl
          page="3"
          writeEnable="true"/>
      </macAcceptanceEntry>
    </managementInterface>
  </device>
  <device
    xsi:type="topo:EndSystem"
    name="upper_floor_node_2"
    syncRole="syncMaster"
    refSyncPriority="#//@syncDomain/@syncPriority[name='syncPriority_1']"
    deviceTarget="TTE_PMC_ESys_1G">
    <port
      name="upper_floor_node_2_P1"
      type="P1"/>
    <port
      name="server_PMGMT"
      type="PMGMT"/>
    <port
      name="server_PSYNC"
      type="PSYNC"/>
    <port
```

```
      name="server_PHOST"
      type="PHOST">
      <macInterface
        name="MAC_INF_0_for_server"
        address="02:02:02:02:FF:44"/>
    </port>
  </device>
  <device
    xsi:type="topo:EndSystem"
    name="basement_node_1"
    syncRole="syncMaster"
    refSyncPriority="#//@syncDomain/@syncPriority[name='syncPriority_1']"
    deviceTarget="TTE_PMC_ESys_1G">
    <port
      name="basement_node_1_P1"
      type="P1"/>
    <port
      name="audioclient_PMGMT"
      type="PMGMT"/>
    <port
      name="audioclient_PSYNC"
      type="PSYNC"/>
    <port
      name="audioclient_PHOST"
      type="PHOST">
      <macInterface
        name="MAC_INF_0_for_audioclient"
        address="02:02:02:02:FF:84"/>
    </port>
  </device>
  <physicalLink
    name="L1"
    transmissionSpeed="1000Mbps"
    mediaType="default"
    refPort="#//@device[name='upper_floor_node_1']/@port[name='
      upper_floor_node_1_P1'] #//@device[name='sw0']/@port[name='sw0_P1']"/>
  <physicalLink
    name="L2"
    transmissionSpeed="1000Mbps"
    mediaType="default"
    refPort="#//@device[name='upper_floor_node_2']/@port[name='
      upper_floor_node_1_P1'] #//@device[name='sw0']/@port[name='sw0_P2']"/>
  <physicalLink
    name="L3"
    transmissionSpeed="1000Mbps"
    mediaType="default"
    refPort="#//@device[name='basement_node_1']/@port[name='
      basement_node_1_P1'] #//@device[name='sw0']/@port[name='sw0_P3']"/>
  <period
    name="sensor_default_period"
    time="100us"></period>
  <period
    name="valves_default_period"
    time="26ms"/>
  <virtualLink
    xsi:type="vl:TTVirtualLink"
    name="vl_valves_cat_1"
    vlid="0"
    refReceivers="#//@device[name='upper_floor_node_1']/@port[name='
      upper_floor_node_1_P1']"
    refSender="#//@device[name='upper_floor_node_2']/@port[name='
      upper_floor_node_2_P1']"
    redundancyMgmt="tt_redundancy"
    refPeriod="#//@period[name='valves_default_period']"
    maxFrameSize="1200"/>
  <virtualLink
    xsi:type="vl:TTVirtualLink"
    name="vl_valves_cat_2"
    vlid="1"
    refReceivers=""
    refSender=""
    redundancyMgmt="tt_redundancy"
    refPeriod="#//@period[name='valves_default_period']"
    maxFrameSize="768"/>
  <virtualLink
    xsi:type="vl:TTVirtualLink"
    name="vl_valves_cat_3"
    vlid="2"
    refReceivers="#//@device[name='upper_floor_node_1']/@port[name='
      upper_floor_node_1_P1']"
    refSender="#//@device[name='basement_node_1']/@port[name='
      basement_node_1_P1']"
    redundancyMgmt="tt_redundancy"
    refPeriod="#//@period[name='valves_default_period']"
    maxFrameSize="1300"/>
  <virtualLink
    xsi:type="vl:TTVirtualLink"
    name="vl_sensors_cat_2"
    vlid="3"
    refReceivers="#//@device[name='upper_floor_node_2']/@port[name='
      upper_floor_node_2_P1'] #//@device[name='basement_node_1']/@port[name='
      basement_node_1_P1']"
    refSender="#//@device[name='upper_floor_node_1']/@port[name='
      upper_floor_node_1_P1']"
    redundancyMgmt="tt_redundancy"
    refPeriod="#//@period[name='sensors_default_period']"
    maxFrameSize="1400"/>
  <virtualLink
    xsi:type="vl:TTVirtualLink"
    name="vl_sensors_cat_1"
    vlid="4"
    refReceivers=""
    refSender=""
    redundancyMgmt="tt_redundancy"
    refPeriod="#//@period[name='sensors_default_period']"
    maxFrameSize="768"/>
```

```xml
  <bestEffortLink
    xsi:type="vl:BestEffortLink_UNDI"
    name="BE_statistics_msg"
    refReceiversMACInterfaces="#//@device[name='server']/@port[name='
      server_PHOST']/@macInterface[name='MAC_INF_0_for_server']"
    refSenderMACInterface="#//@device[name='audioclient']/@port[name='
      audioclient_PHOST']/@macInterface[name='MAC_INF_0_for_audioclient']"
    channel="1"/>
  <bestEffortLink
    xsi:type="vl:BestEffortLink_UNDI"
    name="BE_video_msg"
    refReceiversMACInterfaces="#//@device[name='videoclient']/@port[name='
      videoclient_PHOST']/@macInterface[name='MAC_INF_0_for_videoclient']"
    refSenderMACInterface="#//@device[name='server']/@port[name='server_PHOST
      ']/@macInterface[name='MAC_INF_0_for_server']"
    channel="1"/>
  <constraint
    xsi:type="c:TransmissionDurationConstraint"
    name="max_trans_dur_constraint_for_vl_sensors"
    refVirtualLinks="#//@virtualLink[name='vl_sensors_cat_1'] #//@virtualLink
      [name='vl_sensors_cat_2']"
    maxTimeSpan="2000000ns"/><constraint
    xsi:type="c:TransmissionDurationConstraint"
    name="max_trans_dur_constraint_for_vl_sensors"
    refVirtualLinks="#//@virtualLink[name='vl_valves_cat_1'] #//@virtualLink[
      name='vl_valves_cat_2'] #//@virtualLink[name='vl_valves_cat_3']"
    maxTimeSpan="1000000ns">
</constraint>
</ns5:NetworkDescription>
```

**Listing 8.2:** TTEthernet Network Description [XML]

## 8.4 Generated SCADE file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Initialized by TTE-Scade Generator with the following document: -->
<!-- Mapping:          m_example_1 -->
<!-- Node Name:        upper_floor_node_1 -->
<!-- Document Version: 1 -->
<!-- Content Version:  1 -->
<!-- Contact:          Valentin Ecker -->
<File xmlns="http://www.esterel-technologies.com/ns/scade/1"
    xmlns:ed="http://www.esterel-technologies.com/ns/scade/pragmas/editor/1"
    xmlns:kcg="http://www.esterel-technologies.com/ns/scade/pragmas/codegen/1">
  <declarations>
    <Operator kind="node" name="upper_floor_node_1">
      <inputs>
        <Variable name="s_valve1">
          <type>
            <NamedType>
              <type>
```

```
                <TypeRef name="int"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve2">
          <type>
            <NamedType>
              <type>
                <TypeRef name="int"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve3">
          <type>
            <NamedType>
              <type>
                <TypeRef name="int"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <outputs>
        <Variable name="s_sensor1">
          <type>
            <NamedType>
              <type>
                <TypeRef name="real"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <inputs>
        <Variable name="s_valve4">
          <type>
            <NamedType>
              <type>
                <TypeRef name="int"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve5">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve6">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve7">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <inputs>
        <Variable name="s_valve8">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </inputs>
      <outputs>
        <Variable name="s_sensor2">
          <type>
            <NamedType>
              <type>
                <TypeRef name="real"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <outputs>
```

```xml
        <Variable name="s_sensor3">
          <type>
            <NamedType>
              <type>
                <TypeRef name="real"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <outputs>
        <Variable name="s_sensor4">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <outputs>
        <Variable name="s_sensor5">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <outputs>
        <Variable name="s_sensor6">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
      <outputs>
        <Variable name="s_sensor7">
          <type>
            <NamedType>
              <type>
                <TypeRef name="bool"/>
              </type>
            </NamedType>
          </type>
        </Variable>
      </outputs>
    </Operator>
  </declarations>
</File>
```

**Listing 8.3:** SCADE file format [XML]

## 8.5 Generated Gluecode

```
/*****************************************************************/
/* This file is generated. Do not modify. */
/* Mapping:                   m_example_1 */
/* Node:                      upper_floor_node_1 */
/* SignalDB Document Version: 1 */
/* SignalDB Content Version:  1 */
/* Contact:                   Valentin Ecker */
/*****************************************************************/


#include "tte_interface.h"

/*************/
/*  DEFINES  */
/*************/

/**********/
/*  MAIN  */
/**********/

(void)scade_task_main(void* task_data)
{

    /* allocation of SCADE IOs */
    inC_upper_floor_node_1 upper_floor_node_1_in_variable;
    outC_upper_floor_node_1 upper_floor_node_1_out_variable;

    /* allocation of frames - OUTGOING */

    /* alloc for vl_id:nr vl_sensors_cat_2:3 */
    uint8_t[4] frame_vl_sensors_cat_2_t;

    /* allocation of frames - INCOMING */

    /* alloc for vl_id:nr vl_valves_cat_1:0 */
    uint8_t[3] frame_vl_valves_cat_1_t;

    /* alloc for vl_id:nr vl_valves_cat_3:2 */
    uint8_t[5] frame_vl_valves_cat_3_t;


    /* allocation of TTE Channel handle */
    tte_channel_t tte_channel;

    /* receive and unpack for vl_id:nr vl_valves_cat_1:0 */
    tte_channel = tte_get_channel(0);
    tte_receive(tte_channel, &frame_vl_valves_cat_1);
    upper_floor_node_1_in_variable.s_valve1 =
        (kcg_int)(uint8_t)frame_vl_valves_cat_1[0];
    upper_floor_node_1_in_variable.s_valve2 =
        (kcg_int)(uint16_t)frame_vl_valves_cat_1[1];

    /* receive and unpack for vl_id:nr vl_valves_cat_3:2 */
    tte_channel = tte_get_channel(2);
    tte_receive(tte_channel, &frame_vl_valves_cat_3);
    upper_floor_node_1_in_variable.s_valve4 =
        (kcg_int)(uint32_t)frame_vl_valves_cat_3[0];
    upper_floor_node_1_in_variable.s_valve5 =
        (kcg_bool)(((frame_vl_valves_cat_3[4])/1)|1);
    upper_floor_node_1_in_variable.s_valve6 =
        (kcg_bool)(((frame_vl_valves_cat_3[4])/2)|1);
    upper_floor_node_1_in_variable.s_valve7 =
        (kcg_bool)(((frame_vl_valves_cat_3[4])/4)|1);
    upper_floor_node_1_in_variable.s_valve8 =
        (kcg_bool)(((frame_vl_valves_cat_3[4])/8)|1);

    /* call main SCADE function */
    upper_floor_node_1(&inC_upper_floor_node_1, &outC_upper_floor_node_1);

    /* pack and send for vl_id:nr vl_sensors_cat_2:3 */
    frame_vl_sensors_cat_2[0] =
        (uint16_t)(upper_floor_node_1_in_variable.s_sensor1);
    frame_vl_sensors_cat_2[2] =
        (uint16_t)(upper_floor_node_1_in_variable.s_sensor2);
    if(upper_floor_node_1_in_variable.vl_sensors_cat_2) {
        frame_vl_sensors_cat_2[2] |= ((uint8_t)(1); }
    else{ frame_vl_sensors_cat_2[2] &= ~(((uint8_t)(1)); }
    if(upper_floor_node_1_in_variable.vl_sensors_cat_2) {
        frame_vl_sensors_cat_2[2] |= ((uint8_t)(2); }
    else{ frame_vl_sensors_cat_2[2] &= ~(((uint8_t)(2)); }
    if(upper_floor_node_1_in_variable.vl_sensors_cat_2) {
        frame_vl_sensors_cat_2[2] |= ((uint8_t)(4); }
    else{ frame_vl_sensors_cat_2[2] &= ~(((uint8_t)(4)); }
    if(upper_floor_node_1_in_variable.vl_sensors_cat_2) {
        frame_vl_sensors_cat_2[2] |= ((uint8_t)(8); }
    else{ frame_vl_sensors_cat_2[2] &= ~(((uint8_t)(8)); }

    tte_channel = tte_get_channel(3);
    tte_send(tte_channel, &frame_vl_sensors_cat_2);
}
```

**Listing 8.4:** generated Gluecode [C]

# Bibliography

[1] TTTech Computertechnik AG. *TTEthernet Specification. Version 0.9.1.*

[2] Marte overview. `http://www.omgmarte.org/node/2`. [Online; accessed 01-September-2011].

[3] C. André, F. Mallet, and R. De Simone. Modeling time (s). *Model Driven Engineering Languages and Systems*, pages 559–573, 2007.

[4] E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(12):1217–1229, 1998.

[5] OMG. *OMG Systems Modeling Language (OMG SysMLTM)*, 2010.

[6] T. Weilkiens. *Systems engineering with SysML/UML.* Elsevier/Morgan Kaufmann OMG Press, 2008.

[7] M. Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, 2006.

[8] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[9] F.X. Dormoy. Scade 6: a model based solution for safety critical software development. In *Proceedings of the 4th European Congress on Embedded Real Time Software (ERTS8)*, pages 1–9, 2008.

[10] A. Le Guennec and B. Dion. Bridging UML and safety-critical software development environments. In *Int. Conf. on Embedded and Real-Time Software, ERTS*, 2006.

[11] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. *ACM SIGPLAN Notices*, 38(7):153–162, 2003.

[12] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[13] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating discrete-time simulink to lustre. In *Embedded Software*, pages 84–99. Springer, 2003.

[14] D. de Niz. Diagrams and languages for model-based software engineering of embedded systems: Uml and aadl.

[15] P.H. Feiler. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.

[16] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues. Ocarina: An environment for aadl models analysis and automatic code generation for high integrity applications. *Reliable Software Technologies–Ada-Europe 2009*, pages 237–250, 2009.

[17] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. From the prototype to the final embedded system using the ocarina aadl tool suite. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(4):1–25, 2008.

[18] P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. The topcased project: a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS)*, 2006.

[19] B. Berthomieu, J.P. Bodeveix, C. Chaudet, S. Dal Zilio, M. Filali, and F. Vernadat. Formal verification of aadl specifications in the topcased environment. *Reliable Software Technologies–Ada-Europe 2009*, pages 207–221, 2009.

[20] T. Henzinger, B. Horowitz, and C. Kirsch. Giotto: A time-triggered language for embedded programming. In *Embedded Software*, pages 166–184. Springer, 2001.

[21] S. Poledna and G. Kroiss. The time-triggered communication protocol ttp/c. *Real-Time Magazine*, 4(98):100–102, 1998.

[22] K. Hoyme and K. Driscoll. Safebus. In *Digital Avionics Systems Conference, 1992. Proceedings., IEEE/AIAA 11th*, pages 68–73. IEEE, 1993.

[23] J. Berwanger, R. Mores, P. Fuhrmann, WO Budde, M. Rausch, A. Krueger, G. Hay, F. Bogenberger, M. Sprachmann, D. Millinger, et al. Flexray–the communication system for advanced automotive control systems. *SAE Technical Paper*, pages 01–0676, 2001.

[24] B. Pickles. Avionics full duplex switched ethernet (afdx). *SBS Technologies, May*, 2006.

[25] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The time-triggered ethernet (tte) design. 2005.

[26] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 330–335. IEEE, 2011.

[27] Yaml specifications. `http://www.yaml.org/spec/1.2/spec.html`. [Online; accessed 16-May-2012].

[28] Aadl partners. `http://www.aadl.info/aadl/currentsite/team/index.html`. [Online; accessed 20-December-2011].

[29] Hudak Feiler, Gluch. *The Architecture Analysis & Design Language (AADL): An Introduction*, February 2006.

[30] X. Blanc, M.P. Gervais, and P. Sriplakich. Model bus: Towards the interoperability of modelling tools. *Model Driven Architecture*, pages 900–900, 2005.