DISSERTATION

Executable Time-Triggered Model (E-TTM) for the development of safety-critical embedded systems

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines
Doktors der technischen Wissenschaften unter der Leitung von

O. Univ.-Prof. Dr. Hermann Kopetz
Institut für Technische Informatik 182

eingereicht an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik

von

Juan Martin Perez Cerrolaza
Matr.-Nr. 0627534
Laimgrubengasse 12A/12, A-1060 Wien

Wien, im Dezember 2010 .....

Executable Time-Triggered Model (E-TTM) for the development of safety-critical embedded systems

The development of distributed safety-critical and real-time embedded systems that must satisfy a certain set of timing constraints with an ever-increasing functionality leads to a considerable complexity growth. For example, a current premium car implements about 270 functions that a user interacts with, deployed over 67 independent embedded platforms, amounting to about 65 megabytes of binary code. Tackling the complexity challenge, providing a consistent notion of time and preserving time properties and constraints throughout the development process are key challenges for the development of such systems.

The Executable Time-Triggered Model (E-TTM) is a novel approach for the executable modeling of safety-critical embedded systems based on the Time-Triggered Architecture (TTA), which provides a consistent notion of time based on the sparse-time concept. Different mechanisms are supported for the description of time properties and constraints, which are intrinsically preserved through model refinement steps and execution. For example, in the simulation of periodic control applications, period and phase relationships are kept constant but independently configurable enabling the simulation of models faster, slower or at the same pace as physical time but always producing the same results at the same simulation time instants. The E-TTM also provides additional strategies to tackle the complexity challenge such as abstraction, partition and segmentation.

E-TTM meta-model has been implemented as a C++ library that extends SystemC with the time-triggered Model of Computation (MoC), and enables the codesign and execution of E-TTM models in SystemC. This approach might be used from the early stages of a development process, in order to develop time-triggered executable specifications and Platform Independent Models (PIM). Whenever the system is implemented in a TTA based system, time related properties and constraints might be preserved from the model down to the final implementation.

Executable Time-Triggered Model (E-TTM) for the development of safety-critical embedded systems

Die Entwicklung verteilter sicherheitskritischer eingebetteter Systeme mit ständig wachsender Funktionalität, die bestimmte Echtzeitvorgaben erfüllen müssen, führt zu einer erheblichen Steigerung der Komplexität. Die Komplexität unter Beibehaltung der zeitlichen Zusammenhänge zu reduzieren, sowie die zeitlichen Eigenschaften und Vorgaben im gesamten Entwicklungsprozess zu wahren, stellt eine der Hauptaufgaben in der Entwicklung dieser Systeme dar.

Das vorgestellte ausführbare zeitgesteuerte Modell (E-TTM, Executable Time-Triggered Model) stellt eine neue Herangehensweise dar, um Time-Triggered-Architektur- (TTA)-basierte Systeme zu modellieren und dabei mittels sparse-time base unter Wahrung der zeitlichen Zusammenhänge die Komplexität zu bewältigen. Dieser Ansatz, in dem das Simulationsintervall konfigurierbar ist, die Phasenbeziehungen jedoch erhalten bleiben, erlaubt die Simulation der Modelle mit höherer oder geringerer Geschwindigkeit sowie in Echtzeit. Die hierbei erzielten Ergebnisse stimmen zu jedem Simulationszeitpunkt überein. Darüber hinaus stellt E-TTM weitere Strategien bereit, um die Komplexität zu reduzieren: Abstraktion, Partitionierung, Segmentierung und sparse-time.

E-TTM wurde als C++-Bibliothek implementiert, die SystemC um das time-triggered Model of Computation (MoC) erweitert. Dieser Ansatz kann bereits in der Anfangsphase von Entwicklungsprozessen angewandt werden, um ausführbare TTA-Spezifikationen und plattformunabhängige Modelle (PIM, Platform Independent Model) zu entwickeln. Wann immer ein System als TTA implementiert ist, bleiben zeitliche Eigenschaften und Randbedingungen von der Spezifikation oder dem Modell bis zur Implementierung erhalten.

Acknowledgments

This thesis was carried out during my employment at Ikerlan-IK4 technology research centre within the electronics knowledge area. There are many people who have contributed to the success of this thesis.

First of all I would like to thank Prof. Hermann Kopetz, the head of the Institute für Technische Informatik at the TU Wien, for giving me the opportunity to do a PhD at TU Wien under his direction on the topic of safety-critical embedded systems. The excellent working and creative environment at the institute was only topped by the discussions with Roman Obermaisser, Christian El Salloum and Bernard Huber among others who have been inspiring colleagues in the Real-Time Systems Group.

Many thanks to Ikerlan-IK4 for giving me the opportunity to do a PhD in TU Wien aligned with the research activities of my working group. I would like to thank the scientific directors Javier Mendigutxia and Ana Martinez for giving the required support. Special thanks go to Antonio Perez and many other colleagues who have contributed to the development of this research.

Many thanks to Eugenio Villar and his team for his hospitality at the University of Cantabria, for his contributions, support and reviewing work.

I am lucky to have a family and friends who always support me in what I am doing. Thanks to my wife Aran, my son Unax, my parents Carmen and Gumersindo and my sister Itziar. This thesis is dedicated to my wife, because without her continuous support this thesis would not have been possible.

Contents

List of Figures

List of Tables

Simplicity does not precede complexity, but follows it

Alan Perlis

Chapter 1

Introduction

1.1 Overview

The development of distributed real-time and safety-critical embedded systems that must satisfy a certain set of timing constraints with an ever increasing functionality leads to a considerable complexity growth [Kop07, PC06]. For example, a current premium car implements about 270 functions that a user interacts with, deployed over 67 independent embedded platforms, amounting to about 65 megabytes of binary code [SS04, PBKS07]. Tackling the complexity challenge [Kop07, JTM07, PC06], providing a consistent notion of time and preservation of properties through the development process [HS07, JTM07, BFLS01] are key challenges for the development of such systems.

State-of-the-art models and methods described in Chapter 6 provide different solutions of interest for the design of distributed real-time and safety-critical embedded systems, where multiple models and methods might be used in conjunction during the development process (e.g., UML [wwwj], SysML [wwwf], MARTE [omg08], Matlab / Simulink [wwwe], SCADE [wt], Giotto [Tem05], TDL [Tem05], TMO [MHJGK$^+$00], etc.). However, as identified in this chapter, systematic preservation of time properties from the model down to the implementation is a key challenge that still needs to be addressed in the development of safety-critical embedded systems. The Time-Triggered Architecture (TTA) provides a validated and certifiable core technology for the development of safety-critical embedded systems [JSPP04], based on the sparse-time consistent notion of time [Kop98], that could be used as the foundation of a modeling approach for the purpose of preserving time properties through the model refinement steps down to the implementation. However, available time-triggered modeling approaches (e.g., TMO, TDL) have some

differences with respect to the time-triggered Model of Computation (MoC) [Kop98] that limit their applicability, e.g., different notion of time not based on the sparse-time concept.

This dissertation defines the Executable Time-Triggered Model (E-TTM), a novel approach for the executable distributed modeling of safety-critical embedded systems based on the Time-Triggered Architecture (TTA). This approach might be used from the early stages of a development process, in order to develop time-triggered executable specifications and Platform Independent Models (PIM). The E-TTM meta-model has been implemented as a C++ library that extends SystemC with the time-triggered MoC, and enables the codesign and execution of E-TTM models in SystemC. Distributed models are connected using physical communication channels such as Ethernet and Time-Triggered Ethernet (TTE).

The E-TTM foundation relies on the time-triggered MoC, which provides a consistent notion of time based on the sparse-time concept, and provides different mechanisms to describe time properties and constraints. Time properties and constraints are intrinsically preserved throughout the modeling refinement process, and might be preserved down to the final implementation if the system is based on the TTA. For example, in the simulation of periodic control applications, simulation time period and phase relationships are kept constant but independently configurable, enabling the simulation of models to be executed faster, slower or at the same pace as physical time (in a similar way as video-recorder might be played faster) but always producing the same results at the same simulation time instants.

SystemC is the selected modeling language because it provides multiple advantages. First of all it is an standard language [iee05] that is becoming de facto standard in industrial codesign [BMS08] and has already been evaluated as a suitable modeling language for embedded systems. It is a C++ library that inherits the benefits of object-oriented programming and can be integrated in different design flows and design environments, including constrained transformations from UML to SystemC [YXGB+06] and SystemC to VHDL. Based on this the design and dependability assessment by means of simulation can be done in a single codesign framework. In addition to this, it has already been evaluated as a suitable modeling language for Simulated Fault Injection (SFI) of safety-critical embedded systems [PAAP10] where the global simulation time can be used to define fault injection time instants - durations and trigger simultaneous faults in modules.

1.2 Objective

The objective of this thesis is the 'definition of an executable PIM modeling approach (meta-model) for the design of TTA based safety-critical embedded systems' with the following properties / features:

1. Time and value domain determinism
2. Period-phase conserving simulation
3. Support strategies that tackle the complexity challenge

This objective targets previously identified key challenges for the design of safety-critical embedded systems:

- Consistent notion of time: The E-TTM foundation relies on the time-triggered MoC, which provides the sparse-time consistent notion of time.
- Preserving time properties: Time determinism is required to ensure the preservation of time properties by means of fundamental modeling attributes (composability, consistency and predictability). The period-phase conserving simulation enables the design space exploration of different period configurations, where simulations coupled with the physical time can be executed faster, slower or at the same pace as the physical time but always produce the same results at the same phase time instants.
- Tackle complexity challenge: Determinism is a sufficient precondition for logical reasoning and a solid foundation to tackle the complexity challenge by means of supported abstraction, partition and segmentation techniques [Kop08a].

1.3 Contributions

All in all, the main contributions of this thesis are:

- Definition of a time and value domain deterministic and period-phase conserving executable PIM modeling approach (meta-model) for the design of TTA based safety-critical embedded systems.
- Extend SystemC to support the defined meta-model and required execution framework, which supports single or distributed model simulations coupled or decoupled from physical time

- Tackle complexity challenge by means of abstraction, partition, segmentation and sparse-time.

As an example, the main contributions of this thesis can be applied in the development of safety-critical embedded systems as follows:

- Executable models enable early dependability assessments that reduce the risk of late discovery of safety related design pitfalls. In addition to this, the overall time required for each assessment can be reduced by a proper model partition and distributed execution of the model. This is illustrated in a case study (see Section 7.2).

- The international safety standard IEC-61508 highly recommends fault injection techniques in all steps of the development process [PAAP10]. E-TTM provides time determinism and period-phase conserving simulation that enables the injection of faults at precise time instants. Based on this, Simulated Fault Injection (SFI) can be used in all steps of the design process, in order to analyze the reaction of the system in a faulty environment and to validate the correct implementation of fault tolerance mechanisms. This is illustrated in a case study (see Section 7.3).

- Executable models can be used as reference models for the design and verification, where developed system / subsystems should exchange messages with the same content and at the same time instants as the reference model. Developed system / subsystems can be verified against the single / distributed model executed coupled with the physical time. In a distributed simulation topology, one or multiple model partitions (submodels) could be replaced by developed subsystems and from the message interchange perspective should not be possible to distinguish between submodels and subsystems.

1.4 Structure of this Thesis

This thesis is organized as described below and shown in Figure 1.1:

- Chapter 2 introduces the background and basic concepts on which the work in this thesis is based.

- Chapter 3 describes the proposed Executable Time-Triggered Model (E-TTM) modeling approach that is elaborated in detail in Chapters 4 and 5:

  - Chapter 4 analyses the concept of time with a focus on safety-critical and real-time embedded systems, identifying suitable time representations, time constraints expressiveness and time abstraction models.
  - Chapter 5 describes the E-TTM meta-model (mmE-TTM), the rules and constructs according to which a E-TTM model is created, and the meta-model implementation in SystemC that provides the E-TTM execution framework (xfE-TTM).

- Chapter 6 describes related work to the topic of this thesis, covering synchronous languages, Timing Definition Language (TDL), Modeling and Analysis of Real-Time Embedded Systems (MARTE), Time-triggered Message-triggered Object (TMO), IEC-61499 and Periodic Finite State Machine (PFSM).

- Chapter 7 describes two example case study E-TTM models.
  - A safety-related real-time industrial control application, Voltage / Frequency (V/F) control of an IGCT based multi-megawatt evaluation platform.
  - A safety-critical odometry subsystem integrated within a European Train Control System (ETCS), an on-board automatic train protection system (SIL-4) that protects the high-speed train by continuously supervising the traveled distance and speed.

- Chapter 8 discusses the key results, conclusions of the work presented and an outlook on future research in this area.

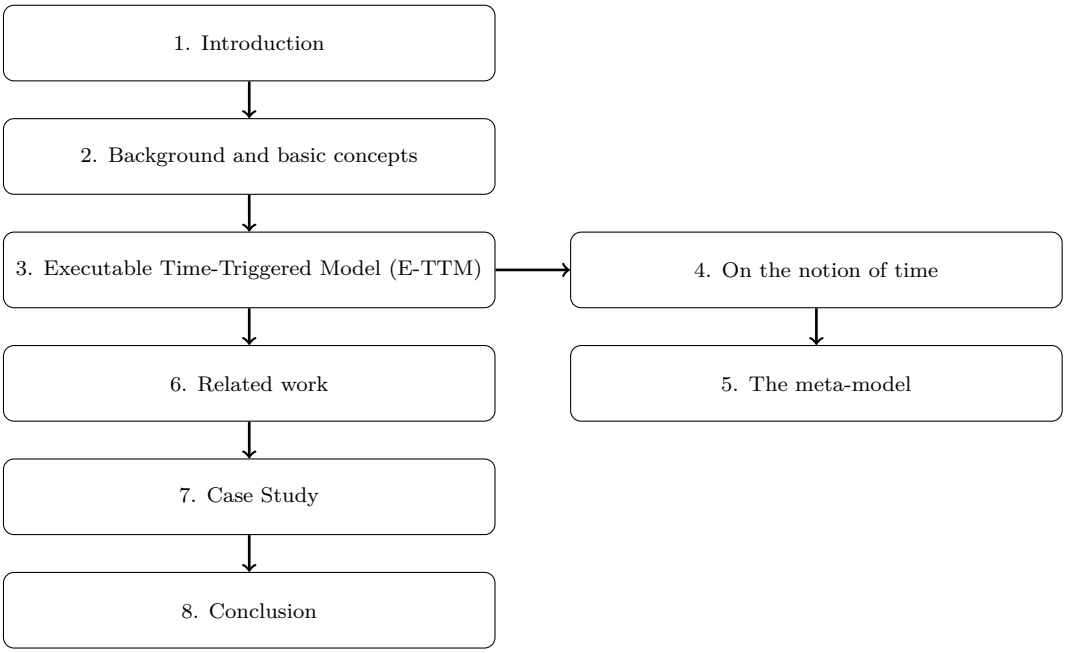


Figure 1.1: Map of chapters.

"Where shall I begin, please your Majesty?" he asked. "Begin at the beginning," the King said, gravely, "and go on till you come to the end: then stop."

Alice's Adventures in Wonderland, Lewis Carroll

Chapter 2

Background and Basic Concepts

The principles used throughout this thesis span over several fields of research. This chapter sets out to introduce the background and basic concepts on which the work in this thesis is based.

2.1 Cognitive complexity

Complexity is defined as "the degree to which a system or component has a design or implementation that is difficult to understand and verify"[dic91] and simplicity is the antonym. The development of distributed safety-critical embedded systems that must satisfy a certain set of constraints (e.g., timing, resources, dependability, etc.) with state-of-the-art technology leads to a considerable complexity growth. For example, a current premium car implements about 270 functions that a user interacts with, deployed over 67 independent embedded platforms, amounting to about 65 megabytes of binary code [SS04, PBKS07]. There is a need to tackle this complexity challenge [RE06, Rum06, Kop08a, JTM07] that might be summarized thus: "One key to achieving dependability at reasonable cost is a serious and sustained commitment to simplicity, including simplicity of critical functions and simplicity in system interactions. This commitment is often the mark of true expertise" [JTM07].

2.1.1 Human cognitive limitations

Complexity and cognitive complexity are used interchangeably within this thesis because complexity is considered only from a cognitive perspective. The cognitive complexity of a task describes the cognitive resources required to

perform this task [Rum06]. Research in this area limits the human cognitive capabilities to the 'magic' number four, up to four simultaneous relationships [RE06, HBMB05] and working memory capacity for up to four simultaneous chunks of information [Cow01]. Relational complexity states that the processing load of a task is determined by the complexity of the relations processed in a given step (the number of independent elements that must be processed simultaneously) [RE06] and it is stated that quaternary relations are the most complex we can handle [RE06, HBMB05]. Human working memory capacity is one of the most limiting factors and it is stated to be limited also to about four chunks of information [Cow01].

2.1.2 Simplification strategies, tackling complexity

Embedded systems might sometimes be unnecessarily complex due to the complexity introduced by the design (accidental complexity), in addition to the complexity which is inherent in the problem being addressed (essential complexity) [Rum06]. Three basic strategies might be used to tackle complexity (in particular accidental complexity) and enable a development to be processed by the limited cognitive capabilities of humans [Kop08a, RE06, Rum06, OK09]:

- **Abstraction** is defined as "a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information" [dic91] so that complexity is reduced by omitting the irrelevant details and focusing on the information relevant for a given purpose.
- **Partitioning**, separation of concerns, deals with the 'spatial decomposition' of a problem into smaller parts that can be analyzed in isolation. The clear separation of computation from communication and the choice of appropriate communication mechanism reduce the complexity.
- **Segmentation** deals with the 'temporal decomposition' of a problem into smaller parts that can be processed sequentially. This reduces the amount of parallel information that must be considered simultaneously.

"A model behaves deterministically if and only if, given a full set of initial conditions (the initial state) at time t_0, and a sequence of future timed inputs, the outputs at any future instant t are entailed" [Kop08a]. Whenever possible, deterministic models (time and value domain) should be developed because they enable logical reasoning, provide proper handling of simultaneity, consistent behavior prediction, higher abstraction levels and also ensure that a properly derived real world implementation will always produce the same result

for the same stimulus in accordance with the original design [GLMS02]. Thus, deterministic models reduce the cognitive complexity and provide predictability of the system. In addition to this, whenever possible use design patterns that make possible the reutilization of proven solutions and proven engineering practices.

2.2 Architecture

There are multiple definitions for the term architecture, from which the following three have been preselected:

1. "The organizational structure of a system or component" [dic00, dic91].
2. "Specific configuration of hardware and software elements in a system" [iec98].
3. "A technical system architecture (or architecture for short) is a framework for the construction of a system for a chosen application domain that provides generic architectural services and imposes an architectural style for constraining an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost effectively" [KOPS04].

Architecting is a consequence of system complexity because it reduces the effort of the design process as the fundamental decisions have already been made [Rum06]. In order to cope with the complexity challenge posed by the development of safety-critical embedded systems [Rum06, Kop07, PC06, Kop97] there seems to be a need for a system foundation that goes beyond the organization / configuration of systems and provides an 'underlying linking structure' (framework) with a set of services and properties. Therefore, the term architecture used within this thesis will refer to the third definition [KOPS04].

TTA (Time-Triggered Architecture)

Time-Triggered Architecture (Time-Triggered Architecture) is a composable and scalable architecture for the design of distributed real-time embedded systems. The Time-Triggered MoC is based on the partition of a large distributed computer system into nearly autonomous systems with small and stable interfaces. The interfaces and predictable time-triggered communication system decouple the interactions among the subsystems from the data processing functions [Kop98]. In TTA, the real-time system is composed of subsystems called

clusters that are composed of nodes interconnected by the *Time-Triggered Protocol* (TTP) real-time communication network [Kop97, KB03, KT98].

- The TTA node is the basic computational building block, which as shown in Figure 2.1 comprises a time-triggered Communication Controller (CC), Communication Network Interface (CNI) and a host processor with memory that executes the operating system and the application software. Clusters are composed of nodes connected to the network by the CC.
- The TTA communication system is autonomous and periodic, the times of periodic fetch and delivery actions are contained in the message-scheduling table, *Message Descriptor List* (MEDL), of each communication controller. The TTA defines three communication protocols [Kop97, KB03, KT98, KAGS05]: the TTP/C is a fault-tolerant time-triggered protocol that provides autonomous fault tolerant message transport, the TTP/A is the low-cost field-bus protocol used to connect low-cost smart transducers to nodes and the TTE (Time-Triggered Ethernet) is a communication infrastructure for integration of real-time and non real-time traffic compatible with Ethernet that supports fault tolerant systems [KAGS05].

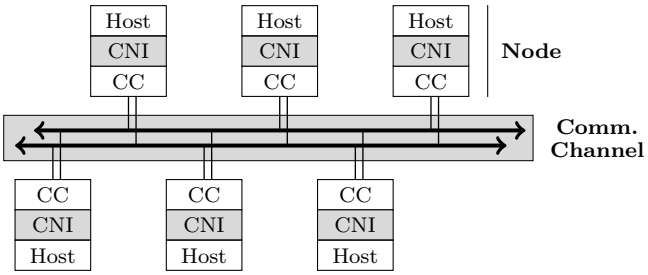


Figure 2.1: Structure of TTA cluster [KB03].

DECOS (Dependable Embedded Components and Systems)

DECOS (Dependable Embedded Components and Systems) [wwwc] is an integrated-architecture that follows a platform-based design based on different abstraction layers as shown in Figure 2.2. DECOS provides technology independent architectural services that serve as a validated stable baseline that reduces application development effort and facilitates reuse [KOPS04, wwwc, MBSP02]. The time-triggered core architecture is the architecture foundation that provides a consistent distributed computing base [MBSP02] that uses time-triggered protocols such as TTP/C and TTE. The core services are the minimal set of services that must be provided across all platforms (predictable message

transport, fault-tolerant clock synchronization, strong fault isolation, and consistent diagnosis of failing nodes). The high-level services are built on top of the core services and used by the application, which might contain multiple DASes (Distributed Application Subsystem), safety-critical (SC) and non safety-critical (NSC).

A *Distributed Application Subsystem* (DAS) is a nearly independent distributed subsystem of a large distributed real-time system that provides a well-specified application service. A DAS can be decomposed into smaller units called jobs that employ ports to communicate with other jobs, and each job has access to its relevant transducers, either directly or via a communication system with known temporal properties. For example, a car system might be divided into the following DASes: steering, braking, powertrain, vehicle dynamics, driver control, passive safety, infotainment, etc. [POT+05].

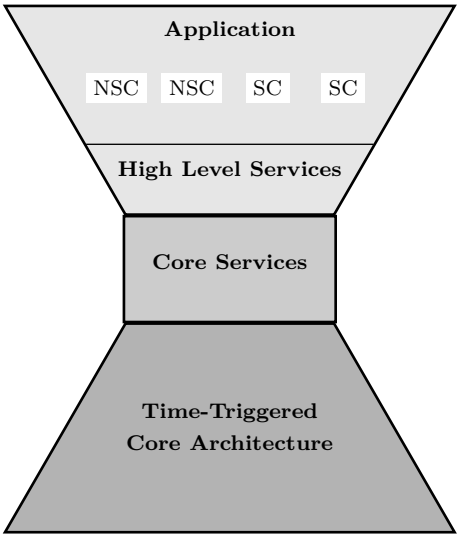


Figure 2.2: DECOS Integrated System Architecture [POT+05]

2.3 Event

Event is defined as "a thing that happens or takes place" [Oxf07] and represents a relevant happening that occurs at a given time instant. Event sequences can be classified as periodic (E_p), sporadic (E_s) and aperiodic (E_a) as shown in Equation 2.1. Periodic event sequences (E_p) are described by an initial offset (r), period (p) and phase (φ). Sporadic event sequences (E_s) are not periodic, but occur at least once within a maximum time interval (a). Aperiodic event sequences cannot usually be defined by a time pattern. From the jobs execution perspective, events are job execution triggering mechanisms that

can be classified into periodic events (time-triggered) and aperiodic / sporadic events (event-triggered).

$$E_p : \forall k \in N_0^+ \rightarrow \exists e : (t_k(e) = r + (k \cdot p + \varphi)) \tag{2.1}$$

$$E_s : \forall k \in N_0^+ \rightarrow \exists e : (t_{k-1}(e) < t_k(e) < t_{k-1}(e) + a) \tag{2.2}$$

$$E_a : unknown \tag{2.3}$$

2.4 Entity

Entity is defined as "a thing with distinct and independent existence" [Oxf07], thus, an entity exists and its existence is guaranteed independent from all other possible entities [Gho99]. Due to its existence, an entity must be self-contained and its behavior defined under all possible scenarios. And due to its independent existence, its behavior is only known to itself and the entities to which it is connected. The concept of entity is widely used in the discipline of digital systems and HDL, where hardware is composed of entities (e.g., AND gate, microprocessor, etc.). Each entity is described by an interface, a behavior description, timing, control, exceptions, etc. [Gho99].

2.5 Component

A component [KOESH07, RE06] is a self contained subsystem that can be used as a building block in the design of a larger system. A component is an entity that provides a desired service to its environment across a well-specified interface, and should maintain its encapsulation (value and temporal) when used in a larger system while encapsulating the inner mechanisms and implementation technology (e.g., software).

Components can be classified into computational components (c-components) and interface components (i-components). Computational components accept input messages, provide a useful service (computation) and produce output messages after some elapsed physical time, i.e., it can contain internal state. Interface components act as gateways, transforming the idiosyncratic representation of the information outside a given cluster into the standardized cluster internal representation [Kop08a].

The precise definition of component interfaces is of utmost importance to reduce the possibility of interactions beyond the control of the designer. Components interact with other components and the outside world through the exchange of messages across interfaces, which as shown in Figure 2.3 four different types of interfaces shall be distinguished [KOESH07, RE06]:

- *Linking Interface* (LIF) is the *Real-Time Service* (RS) interface that offers the services of the component to other components. It is the most important interface from the system integration and composability point of view, because it must provide all the information needed to understand the behavior of the component, while hiding the implementation details.
- *Configuration and Planning* (CP) is the interface for the component configuration.
- *Diagnostic and Maintenance* (DM) is the interface used for maintenance through which it is possible to observe and modify the internal elements of the component.
- Local interfaces are linked to the environment controlled by the component. Closed components contain no local interface to the real world while open components contain a local interface to the real world. Open components might be a source of indeterminism [Kop97].

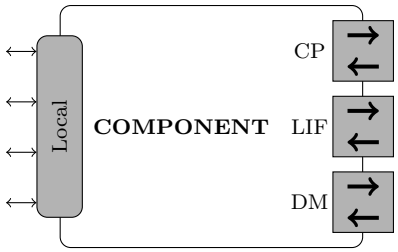


Figure 2.3: Interfaces of a component [KOESH07].

2.6 Model-Based Design (MBD)

Model-Based Design (MBD) is a system design approach supported by different modeling language standards from OMG (e.g., UML [wwwj], SysML [wwwf] and MARTE [omg08]), Matlab / Simulink [wwwe] de facto standards in control systems, SCADE [wt] for safety-critical embedded systems and additional multiple modeling languages of interest (e.g., Giotto-TDL [Tem05]). In MBD the system behavior and functionality is first designed in a *Platform Independent Model* (PIM) and then mapped and refined to a given platform using a *Platform Specific Model* (PSM). An executable high-level model of an application, the PIM, captures and documents the intended application properties (function, timing) of the evolving design without regard to the idiosyncrasies of the targeted execution platform. At a PIM level, a DAS can be represented

by a set of time-aware computational components, PIM-jobs, which exchange messages [KOESH07].

2.6.1 Model

A model is defined as "a simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions" [Oxf07] and the essence of modeling lies in accuracy for the stated purpose, simplification and understandability. In order to reduce the cognitive complexity, a modeling language having the following features is preferable: a small number of well-specified concepts and relationships that focus on the essential properties (e.g., time and value domain), with a clear structure and model functions, with optional formal notation and clearly stated model assumptions. Therefore, models used in the development of embedded systems should be simple and understandable (to reduce cognitive complexity), they should provide simplification strategies based on coherent model assumptions coverage and whenever possible should be deterministic in the time and value domain for the given model assumptions [Kop97, Kop08a].

2.6.2 Meta-Model

A meta-model "is a model of a modeling language" [Kur05] that defines the rules and constructs according to which a model is created, thus, a model is frequently considered to be an instance conforming a meta-model. For example, the UML meta-model describes the UML language, which is used to model systems.

2.6.3 Model transformation

Model transformation is the process of converting one model into another model, frequently using a model transformation language that enables the automatic process of transformation to take place [Kur05, Tru07]. According to Kurtev, there are three types of model transformations [Kur05, Tru07]:

- Refactoring transformation reorganizes a model, based on some precisely defined criteria.
- Model-to-model transformation transforms one model to another model (e.g., PIM to PSM).
- Model-to-code transformations convert models into text, such as source-code (e.g., C++).

2.7 Naming

Naming is an important but frequently overlooked area in the design of distributed embedded systems, as it is for computer science in general [Pas99]. The name provides an unambiguous identifier which specifies a unique entity (uniqueness). Names should not only be meaningful but also reflect the correct intended meaning following a human-oriented conceptual knowledge about the world, so that it enables a person (e.g., the engineer) to identify and understand the functionality and the structure of a given system. These names might be qualitatively different from the names used to construct the systems (e.g., names used in the software source code) [Pas99, DP05, BMW93].

Name scheme

A naming scheme consists of the syntactic and semantic interpretation of names, and the set of names complying with a naming scheme forms a name space. If a uniform name scheme is provided, any system resource of interest can be bound to a name [THH02]. The *Uniform Resource Identifier* (URI) [BL05] was defined by the IETF in order to access abstract and physical resources available via the Internet, and might be classified as a name (URN), a locator (URL) or both.

- The *Uniform Resource Name* (URN) [Moa97, Spa04] is a URI that uses the 'urn scheme' and sets out to be a persistent and location independent resource identifier (e.g., "urn:isbn:0792398947" specifies the book [Kop97]). The URN naming scheme is defined as described in Equation 2.4 [Moa97], where NID is the Namespace Identifier (e.g., ISBN) and NSS is the Namespace Specific String (e.g., 0792398947).
- The *Uniform Resource Locator* (URL) specifies the location of a resource, how it can be retrieved and sometimes the name as well (e.g., http://www.ieee.org).

$$< URN >::= "urn : " < NID > " : " < NSS > \tag{2.4}$$

2.8 SystemC

SystemC [iee05, wwwg, GLMS02] is an open-source C++ class library used to develop executable models of hardware-software systems at different levels of

abstraction, standardized as IEEE-1666 [iee05]. Strictly speaking SystemC is not a language, but it is usually referred as a *System-Level Design Language* (SLDL), and it is not a software design environment but a software-hardware development and execution environment. One of the most important features of SystemC design flow is that both hardware and software components can be described using a common language (codesign of HW and SW). In addition to this, there is an interesting research effort towards the usage of UML and SystemC in system level modeling of the software, where UML is used to design the software and generate the SystemC executable implementation [YXGB+06, NZTWF04].

2.8.1 Time

SystemC simulation time provides a discretized time abstraction model of configurable granularity, from femtoseconds to seconds. The simulation engine updates the simulation time that acts as a virtual physical time for the simulation, decoupled from the real world physical time. The simulation time is represented by a 64 bit-unsigned integer [iee05]. SystemC simulation time might be used to provide a simulation global time of configurable granularity. In addition to this, time deterministic models can be designed if appropriate model design restrictions are applied [GLMS02, iee05].

Delta-delay

SystemC events are instantaneous, they have no value and no duration, and can be established consistently at all simulation processes. The computational time of a process is zero (simulation time) and all processes sensitive to a given event will be triggered in the same consistent discretized point in time in case of event activation, leading to simultaneous-instantaneous execution of processes. In order to execute simultaneous processes, SystemC imposes a partial order for each delta-cycle that lasts for an infinitesimal amount of time and which does not advance the simulation time. That means that all simultaneous actions are executed concurrently in zero simulation time, the delta cycle. SystemC supports the same delta-cycle concept as VHDL [IEE94] and Verilog.

Time determinism

In order to develop time deterministic models (see Section 2.1.2), simultaneity of events must be properly handled [Kop08a]. This requires the following minimum recommendations to be met [iee05, GLMS02, Gho02]:

- Whenever possible use a MoC that is known to guarantee determinism.
- Whenever possible use primitive channels with request-update (e.g., sc_signal) or rely on a two-phase synchronization scheme communication, for inter-module and inter-process communication.
- Avoid synchronization mechanisms that could lead to race-conditions such as mutexes and semaphores.

Clock

The SystemC class sc_clock models hardware clocks that generate periodic events of configurable period and phase. All clock instances are based on the same global simulation time and initialized before simulation starts, thus all clock period and phases are consistent during the simulation.

2.8.2 Simplification strategies, tackling cognitive complexity

SystemC could be used to provide all three previously defined model simplification strategies, described in Section 2.1.2, for the development of safety-critical embedded systems:

- Abstraction: SystemC provides the notion of global time and a powerful functionality description capability based on C++. It provides structural decomposition by means of hierarchical modules which only expose the required interface while the internal implementation details are hidden. *Transaction-Level Model* (TLM) is a high-level approach to modeling embedded systems, were communication between modules is modeled using function calls abstracted from the detailed implementation of the communication architecture [GLMS02].
- Partitioning: SystemC provides a strict separation of computation and communication, and message based communication support.
- Segmentation: The internal functionality of SystemC modules, can be described with other modules and processes that execute sequentially or simultaneously. Simultaneous execution, is performed sequentially by the execution engine using the delta-cycle concept.

2.8.3 Codesign of HW and SW

One of the most important features of SystemC design flow is that both hardware and software components can be described using a common language. In fact, at the beginning of the design both are indistinguishable as the assignment of modules to hardware or software has not yet been made. This enables a seamless exploration of different architectures and hardware vs. software tradeoffs without the need for describing modules in domain specific languages such as C and VHDL [Pan01].

- **Software**: There is an interesting research effort towards the usage of UML and SystemC in system level modeling of the software, where UML is used to design the software and generate the SystemC executable implementation [YXGB+06, NZTWF04]. However, the current SystemC version does not support software tasks and preemptive scheduling, which are scheduled for SystemC version 3.0. Therefore, current support for complete software modeling and execution is limited but looks promising in conjunction with UML.
- **Hardware**: A subset of SystemC models typical hardware functionality by means of constructs analogous to HDLs, so it can synthesized or translated into HDL languages such as VHDL [Pan01].

2.8.4 Extensions

SystemC aims to provide a simple foundation that can be extended to support heterogeneous MoC, design libraries, modeling guidelines, and design methodologies that are required for system design. SystemC currently only supports the discrete-event MoC providing a lightweight event-triggered execution engine with simple and flexible synchronization capabilities provided by events and wait statement. Based on this different extensions can be built on top of SystemC, such as SystemC-AMS [wa] and HetSC [HVG+07, VH08].

SystemC-AMS (SystemC - Analog Mixed Signal) [wa] extends SystemC to support the functional simulation, simulation based verification, Transaction-Level Model and modeling of analog and mixed signals [VGE03b, MDHH05]. SystemC-AMS extension provides support for the modeling and simulation of continuous time-models and heterogeneous models such as mixed-signal (analog and digital) and mixed-domain (e.g., electromechanical) [VGE03b, VGE04]. A synchronization layer, provided as part of the SystemC-AMS library, ensures the timing consistency [VGE03b, VGE03a, HVG+07].

The identification of abstraction levels and the development of corresponding deterministic models, where the indeterminism of the world at the lower levels does only have a negligible effect, are at the root of scientific discovery and engineering practice [Kop97]

HERMANN KOPETZ

Chapter 3

Executable Time-Triggered Model (E-TTM)

This chapter provides an overview of the Executable Time-Triggered Model (E-TTM), which is elaborated upon in detail in the following Chapters 4 and 5. The E-TTM is a time deterministic executable modeling approach for the composable development of safety-critical embedded systems based on Time-Triggered Architecture (TTA). E-TTM might be used from the early phases of a V-model development process, in order to provide executable specifications and PIM models that could also be used during the verification and validation phases. It targets both Platform Independent Model (PIM) and Platform Specific Model (PSM) views, but only the PIM view is elaborated upon within this thesis.

3.1 Introduction

E-TTM provides a component-based modeling approach for systems based on Time-Triggered Architecture (TTA), a time deterministic executable modeling approach for the composable development of safety-critical embedded systems. For this purpose, it provides a consistent notion of time based on the sparse-time concept from the time-triggered MoC [Kop98], expressiveness to describe time properties and constraints, support for replica determinism (e.g., TMR) [PBWB00], mechanisms to tackle the complexity challenge [Kop08a, JTM07], etc. Time properties and constraints are intrinsically preserved through model refinement steps and execution, and through the development process whenever the implemented system is based on the TTA. Nonetheless, E-TTM could also be used for the development of (generic) distributed real-time and safety-critical embedded systems, not necessarily based on the TTA, where the de-

signers are responsible for ensuring that properties of interest are preserved. The E-TTM approach involves a set of constraints that can be relaxed for this purpose by the designers.

The E-TTM simulation time is invariant of the physical time progression and the distributed simulation topology. When modeling periodic controls applications the simulation time might be coupled or decoupled from the physical time. If it is coupled, a periodic time synchronization is performed at every macrotick, but the ratio between simulation time and physical time might be configurable in order to enable the execution of models faster, slower or at the same pace as physical time. This is similar to a video-player which can play films faster, slower or at the same pace as physical time, but in all cases, the executed model provides the same simulation results at the same simulation instants because simulation time periods and phases are kept constant. This simulation time result invariance property is also supported in the execution of distributed simulation topologies.

3.2 Views (PIM and PSM)

In order to reduce the design cognitive complexity, E-TTM supports both Model Driven Architecture (MDA) abstraction views, Platform Independent Model (PIM) and Platform Specific Model (PSM). For each abstraction level the interface description can be partitioned and analyzed in isolation or conjunction, and designers can concentrate on different attributes and properties meaningful for each development stage. This enables the cooperation of different teams through the development stages such as system control team at the PIM level and embedded-systems team at all levels with an emphasis on the PSM level.

- **Platform Independent Model (PIM):** At this level the designer specifies the structure, interfaces (time and value domain), describes the functionality (semantic and syntactic), specifies dependability requirements (e.g., redundancies) and specifies performance constraints that can be refined through the modeling process. The PIM structures the overall application functionality into systems, DASs and jobs [HOP06, KOESH07]. The designer is abstracted from technical details and platform specific requirements.
- **Platform Specific Model (PSM):** At this level the designer maps previous PIM into physical distributed platforms, taking into account platform specific requirements and constraints (e.g., Time-Triggered Network-on-Chip (TTNoC) [OESHK08]).

3.3 V-model life cycle

The development of safety-critical embedded systems usually follows the well known V-model approach shown in Figure 3.1, at least for industrial and transportation control domains [PC06]. The left part of the V-model includes the specification, design and development phases, which as shown in Figure 3.2 could be done in consecutive steps that starting from the 'what' (specification) leads to the 'how' (synthesis, development) using different abstraction views in the design process (PIM, PSM).

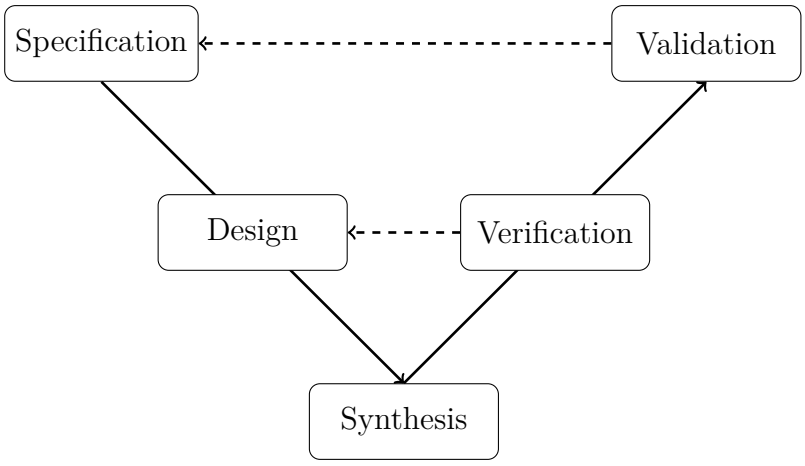


Figure 3.1: Simplified V-Model life cycle.

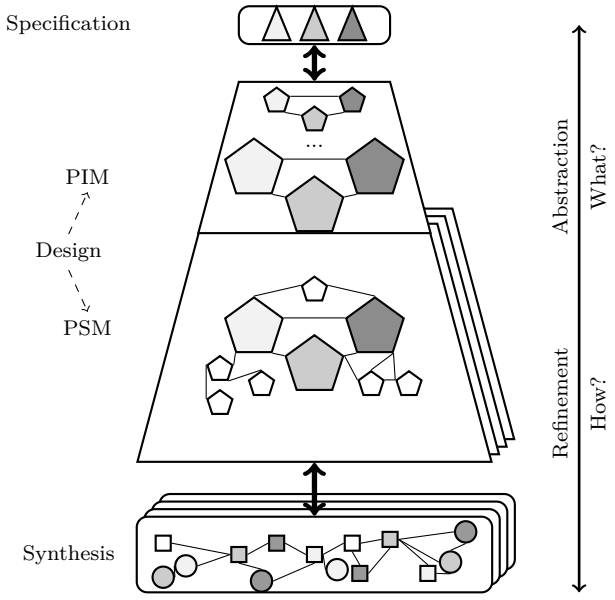


Figure 3.2: Multi-step development process [HVF+05].

The E-TTM modeling approach follows this abstraction and refinement approach and could be used from the early phases of a V-model development process in order to provide executable specifications and PIM models. These models could also be used during the verification and validation phases, for comparison purposes between the specified or PIM modeled system and the developed system. For example, a distributed E-TTM model coupled with the physical time should exchange messages with the same content and at the same time instants as the equivalent physical system should: thus, it would not be possible to distinguish whether the sender of messages is a submodel or a subsystem by just comparing the arrival time and content of messages.

3.4 Meta-model

As shown in Figure 3.3 the E-TTM modeling approach is based on a meta-model definition and implementation, described in Chapter 5. The E-TTM meta-model (mmE-TTM) specifies the rules and constructs according to which a E-TTM model is created. On the other hand, the meta-model implementation provides the meta-model elements required to develop E-TTM models and the meta-model E-TTM execution framework (xfE-TTM) on top of which developed models are executed.

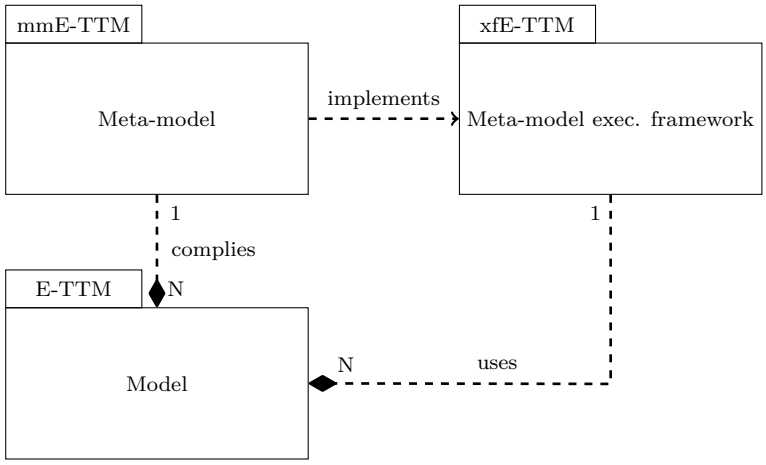


Figure 3.3: E-TTM model, meta-model and execution framework.

3.5 Notation

The following Table 3.1 describes the notation used to describe the E-TTM:

Symbol	Description
et/tt	Event-Triggered / Time-Triggered
i/o	Input / Output
e	Event, e^+ triggered and e^- non triggered
$\mathbb{E}$	Set of events, $\mathbb{E} = \{e\}$
c	Component
$\mathbb{C}$	Set of components, $\mathbb{C} = \{c\}$
$x(c)$	Component (c) execution
$e(c)$	Component (c) activation event
$port$	Interface port, $port \in [LIF, CP, DM]$
m	Message
$\mathbb{M}$	Set of messages, $\mathbb{M} = \{m\}$
s/r	Send / Receive message
Π	Sparse-Time activity interval
Δ	Sparse-Time silence interval
Γ	Sparse-Time macrotick,
$tick$	Global Time tick, $tick \in N^+, tick = K \cdot \Gamma$
δ	Delta delay

Table 3.1: E-TTM notation.

What then is time? If nobody asks me, I know what time is, but if I am asked then I am at a loss what to say.

St. Augustine [J.00]

Chapter 4

On the notion of time

This chapter analyses the notion of time from different perspectives related to the development of safety-critical embedded systems, identifying suitable time representations, time constraints expressiveness and time abstraction models to be included in the definition of the E-TTM.

The concept and nature of time [Dav95, Gal03, J.00, Mer05] has been analyzed by philosophy, art, religion and science for thousands of years but still no common agreement is available among them. The relativity of time and it's implications makes any agreement even more difficult. What St. Augustine (AD 354-430) said several centuries ago still might be considered valid nowadays "What then is time? If nobody asks me, I know what time is, but if I am asked then I am at a loss what to say" [J.00].

4.1 Introduction

The concept of time commonly used in the development of real-time embedded systems is based on the Newtonian physics concept of time, disregarding relativistic effects. Time has a key role in the development of Safety-Critical Embedded-Systems (SCES) and Real-Time Embedded System (RTES), because a notion of physical time is required to express time requirements that must be met by the system (e.g., hard deadline). However, historically the notion of time has not been supported with the required rigour by computer science [Lee99, MCIJ+02, Gho99], as stated by Lee: "Time has been systematically removed from theories of computation, since it is an annoying property that computations take time. 'Pure' computation does not take time, and has nothing to do with time. It is hard to overemphasize how deeply rooted this is in our culture. So called 'real-time' operating systems have so little to go

on that they often reduce the characterization of a component (a process) to a single number, its priority. " [Lee99, MCIJ+02].

4.2 Basic Concepts

Time is defined as "the indefinite continued progress of existence and events in the past, present, and future, regarded as a whole" [Oxf07]. The second is the unit of time according to the International System of Units (SI), defined as "the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom". In addition to this, time is considered to be a fundamental quantity in physics, other units of physical quantities can be generated from it [BIP08].

4.2.1 Flow of time

The flow of time based on Newtonian physics might be modeled as a directed timeline that extends from the past to the future, as shown in Figure 4.1. An *instant* is a cut in the timeline where the instant now, the present, separates the past from the future. An *event* is a relevant happening that occurs at an instant and the *interval* between two instants is called a *duration* [OESHK07]. The timeline might be modeled as dense / continuous (e.g., real world) or discretized.

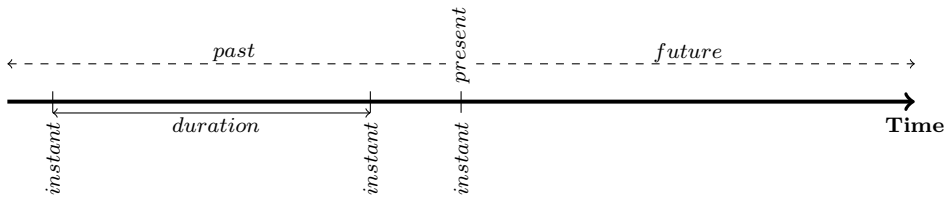


Figure 4.1: Flow of time.

4.2.2 Time measurement

A (physical) clock is a device for measuring time, and almost every clock may be considered a two-part device, an oscillating device for determining the length of a periodic time interval and an increasing counter. The periodic event is called the *microtick* [Kop97] of the clock and the time duration between two consecutive microticks is called the *granularity* of the clock. Atomic clocks are the most accurate clocks, they are claimed to be so accurate that they would

neither gain nor lose even a second in more than 200 million years [LZC+08]. The reference oscillation of an atomic clock is based on an electromagnetic signal associated with a quantum transition between two energy levels in an atom [Kop97, AAH97].

A physical clock can exhibit two failure modes, the counter could be mutilated by a fault and / or the drift rate of the clock could depart from the specified drift rate which for a perfect clock would be zero [Kop97]. In principle, if a clock were set perfectly and the frequency remained perfect, it would keep the correct time indefinitely. But this is not possible for real clocks [AAH97] because it is not possible to set clocks perfectly (due to random and systematic variations intrinsic to any oscillator mechanism), environmental causes can cause the clock frequency to vary from the specified frequency and time relativity (time is a function of position and motion).

GPS is a global navigation system that relies on precise timing technology required for this purpose, based on synchronized atomic clocks that even take the theory of relativity into consideration [Gal03]. Due to the low receiver equipment cost it has become the world's principal supplier of accurate time and might be used by real-time embedded systems as an external clock synchronization source [Kop97, AAH97].

4.2.3 Causality

Causality is a fundamental law in the physical universe and philosophy, which states that "for every cause there is an effect and for every effect there must have been a cause", and timing constitutes the external and observable manifestation of the causal relationship between activities [Gho99].

If multiple causal events happen within a system, the exact temporal order of the events is helpful in order to identify the primary event(s). If event $e2$ occurs after event $e1$, then $e1$ might be the cause of event $e2$ but not otherwise. Therefore, the temporal order of events is necessary but not sufficient in order to establish the causal order [Kop97].

Causes and effects cannot be simultaneous in reality within a dense timeline, zero time interval between the cause event and the effect event, because the speed with which an action may occur is limited by the speed of light and therefore the time interval might be very small (infinitesimal) but always bigger than zero. Therefore, the principle of causality helps to distinguish between the past and the future [Gho99].

4.2.4 Determinism

Determinism [PC07b, Kop08a] means that for a given set of relevant conditions a given item (e.g., property, output, etc.) is completely predictable and does not depend on randomness or stochastic statements. As shown in Equation A.7, and explained in Section A.5 on page 134, this implies that an output (o) is deterministic for a given set of relevant conditions (c), if given the same set of initial conditions then the system (s) always generates the same outputs at the same time (ts_o) when given the same inputs (i) at the same time (ts_i).

This definition implies that the notion of time must be consistent at the system level under analysis, which means for example that in the case of a distributed embedded system the notion of time should be based on the sparse time (see Section 4.5 on page 38). It also means that if the system under analysis and the external observer that makes the judgement do not share the same notion of time, determinism might just be a local property of the system under analysis and not an 'absolute' property from the observers perspective (it is 'relative').

4.2.5 Simultaneous, synchronous and coincident

Simultaneous, synchronous and coincident are three similar concepts that describe the occurrence of an event at about the same time:

- **Synchronous** is defined as "existing or occurring at the same time" [Oxf07] and derives from the Greek words *sun* "together" and *khronos* "time".
- **Simultaneous** is defined as "occurring, operating, or done at the same time" [Oxf07] and derives from the Latin word *simul* that means "at the same time".
- **Coincidence** is defined as "correspondence in nature or in time of occurrence" [Oxf07].

According to their definitions synchronous, simultaneous and coincident might seem to be synonyms but they do in fact represent different concepts. In contrast to simultaneity and synchrony, coincidence expresses the occurrence of events at about the same time (usually within a wide time interval) without implying a relationship or dependency among them. Synchrony is related to the triggering of an event or the repetitive time pattern in which two or more events are activated at the same time instants (or harmonic). On the other hand simultaneous is related to the occurrence of multiple events within a narrow or zero time interval, and this occurrence might not be always repetitive. Synchronous

languages rely on the zero execution time principle, thus executions are both synchronous and simultaneous. However, in the real world synchronous events are not always simultaneous. Imagine for example two parallel Analog Digital Converter (ADC) that sample analog values:

- Synchronous and simultaneous: If the conversion is triggered by the same clock and the ADC conversion time is equal, the conversion will be synchronous and simultaneous (disregarding delays due to different hardware signal routing).
- Synchronous: If the conversion is triggered by the same clock and the ADC conversion time is different (e.g., different types of ADC), the conversion will be synchronous but not simultaneous (not happening at the same time).
- Simultaneous: If the conversion is triggered by different clocks with the same frequency and offset difference equal to the ADC conversion time difference, the conversion will be simultaneous but not synchronous. This could also be the case for different independent ADCs and triggering clocks, that by chance perform a simultaneous conversion (coincidence).

If not handled properly, simultaneity of events might be a challenge to the designer of SCES because it might be a source for indeterminism and unpredictability in the time and value domains: the meta-stability problem in hardware, the mutual exclusion problem in operating systems, the consistent message ordering problem in distributed systems, race conditions, etc. At a distributed embedded system level, simultaneity of events happening in a dense timeline is at the root of indeterministic behaviour (time and value domain) because it is basically impossible to arrive to a system wide consistent notion of simultaneity [Kop08a].

4.2.6 Parallel, concurrent and sequential

Parallel, concurrent and sequential are three related concepts that describe the execution of activities during time:

- **Parallel** is defined as "occurring or existing at the same time or in a similar way" [Oxf07] and derives from the Greek word *parallelos*, *para* meaning "alongside" and *llelos* "one another".
- **Concurrent** is defined as "Existing, happening, or done at the same time" [Oxf07] and derives from the Latin word *concurrent* that means "running together, meeting".

- **Sequential** is defined as "forming or following in a logical order or sequence" [Oxf07]. Sequence is defined as "a particular order in which related things follow each other" [Oxf07] and derives from the Latin word *sequent* that means "following".

Sequential execution implies no simultaneous execution, only one execution at a time and one after another in time. On the other hand, parallel and concurrent might seem to be synonyms but they represent indeed different concepts. Parallel execution of activities imply that activities might be executed simultaneously, which means that at a given point in time several activities might be executed. Concurrent execution of activities might be in parallel, or sequential by interleaving all different activities in time giving the illusion of parallelism.

4.2.7 Periodic, aperiodic and sporadic

Periodic, aperiodic and sporadic are three related concepts that describe the time interval execution of activities or activation of events:

- **Periodic** is defined as "appearing or occurring at intervals" [Oxf07] and derives from the Greek word *periodikos* meaning "coming round at intervals".
- **Aperiodic** is defined as "not periodic; irregular" [Oxf07].
- **Sporadic** is defined as "occurring at irregular intervals" [Oxf07] and derives from the Greek word *sporadikos* meaning "scattered".

Periodic events are cyclic, they occur at fixed known time intervals. Sporadic events can occur at arbitrary points in time, but with defined minimum inter-arrival times between two consecutive events. On the other hand, aperiodic events have irregular time intervals, not known a priori, which might be random or described only by statical means [Kop97].

4.2.8 Instantaneous

The term instantaneous is defined as "occurring or done instantly" [Oxf07], which means that either the execution activity is triggered in the same instant (zero time interval delay) and / or the execution time of an activity is zero (it happens in the same instant, where a time instant has no duration, zero time interval). If the interval of occurrence is (very) small compared to all other intervals of interest, this occurrence might also be considered to be 'instantaneous' even if the interval time is not zero. Instantaneously, like simultaneity, might be a source of non-determinism if not handled properly. In fact, multiple instantaneous events are simultaneous events.

4.3 Cognitive Time

In our reality, at the level of human comprehension, time is synonymous with the wall clock [Gho99]. The human perceptual sense of time is a fundamental cognitive life function according to the Layered Reference Model of the Brain (LRMB), which might be considered as the thinking engine of the brain with a seven-layered hierarchical structure, where time is the fifth [Yin08]. The cognitive capabilities, the mental process by which time is perceived and reasoned (psychology and neuroscience), plays a central role in the design of real-time embedded systems by human designers.

4.3.1 Time perception

The brain measures time continuously and the sense of time is needed for most human life functions such as planning, coordination and execution of activities, observing the sequence of events, etc. Time perception is an ability usually taken for granted, but still relatively little understood and experiments on human time perception are in their infancy. Different studies suggest the involvement of multiple human brain regions in timing functions and the minimum interval of human sense of time is identified within the scope of $25ms$ to $150ms$ [LWS07, Yin08, Dav95, Eag08]. Time perception is slightly different among people and might depend on the context (e.g., frightening situation) [Eag08].

Temporal processing is likely distributed and temporal and spatial processing are intrinsic properties of neural functions [MB04]. However, the neural mechanisms for time perception seem to be different for different timing scales, cognitive time perception deals with 'long scales' (e.g., second, minute, hour, day, month, etc.) and 'automatic' or 'direct sensation' deals with subsecond timing [Eag08, Ram99].

- On the subsecond timing scale, 'automatic' or 'direct sensation', Eagleman states that a diverse group of neural mechanisms seem to mediate temporal judgements. Multiple research results in neuroscience state that the brain represents time in a distributed manner, and provides a single synchronized notion of time by detecting the coincidental activation of different neural populations [BM05, MB04, Eag08]. Temporal judgements are constructions of the brain that enable a consistent reconstruction of the world around us (e.g., causality), but can be twisted in laboratory experiments converting them into temporal illusions [Eag08].
  - For example, different sensorial information requires different processing time in the brain, e.g., hearing is processed faster than visual

information. However, the brain synchronizes all sensorial information by delaying some of it in order to produce a consistent reconstruction of our 'reality', so that for example the visual image of a person clapping and the hearing of the clapping is simultaneous. Even if the person is relatively far away from us, thus the clapping noise will physically arrive later (speed of sound vs. speed of light), the brain will still try to synchronize this information and it will seem to be simultaneous to the human observer if the delay of the arriving information is below approximately 1/10th of a second.

- On a larger time scale (bigger or equal to one second), there seems to be a single source of temporal information (single internal abstract clock) used by the cognitive system [vRT08]. This is analyzed in the following Sections.

4.3.2 Asynchronous brain, multiple clocks

The human sense of time can be described by the physical, cognitive and biological clocks as shown in Figure 4.2.

- **The cognitive clock** is a conscious and subjective perception of time based on the internal biological clock and a conscious relative perception of the external physical clock.
- **The biological clock** is an unconscious and subjective notion of time based on the physiological and biological rhythms of the human body. The typical pacemakers of the biological rhythms, the subconscious ticks, are the sleep-awake cycle, heartbeats, breathing, metabolic activities, etc. The sense of time can be altered by multiple biological and physiological factors (e.g., the biological pace is proportional to the temperature of the human body) [Yin08].

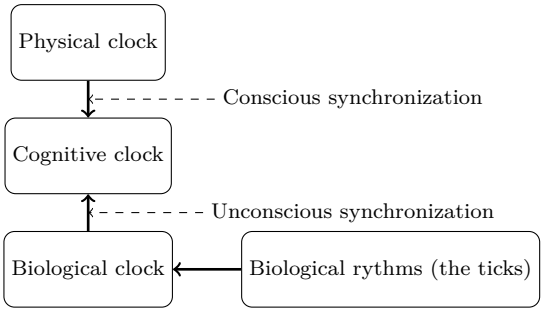


Figure 4.2: Sense of time according to the internal and external clock [Yin08].

The brain seems to be an asynchronous system, there is no obvious central clock at the conscious, subconscious or physiological levels [LMZO03, Sta05, Yin08, LW05]. However, different research studies have analyzed how humans perceive and reproduce intervals of time relatively accurately and the capabilities for timing multiple overlapping intervals. One of the conclusions of this research is that a single source of temporal information (single internal abstract clock) is used by the cognitive systems to account for the estimation of partly overlapping intervals [vRT08].

In addition to this, different parts of the brain might act as pacemakers of the biological clock, such as the hypothalamus which is believed to be the responsible for the circadian rhythm [Yin08]. There also seems to be a biological clock synchronization with the external physical time to synchronize daily activities with the external world. The physiological rhythmic cycle tends to synchronize at a 25 hours cycle per day, rather than 24 hours, and this might indicate that human beings still keep an ancient rhythmic cycle which was formed several hundred million years ago when the daily cycle of the earth was longer than nowadays [Yin08].

All in all, it seems as if the cognitive clock acts as a notion of global time at the brain level (the single internal abstract clock), based on a transparent internal clock synchronization (e.g., biological clocks, neural populations, etc.) and external physical clock synchronization (e.g., watch, daylight, etc.).

4.3.3 Concurrent multitasking

One of the most impressive aspects of human cognitive capability is concurrent multitasking, the ability to manage and execute multiple concurrent tasks. In some situations the concurrency is ubiquitous and seems to be effortless (e.g., walking and talking), while others seem to be extremely difficult if not impossible (e.g., reading one text and listening to a different one). Threaded cognition deals with concurrent multitasking and aims to provide a theoretical and computational framework for understanding, modeling and predicting performance of the concurrent execution of arbitrary tasks [ST08].

The development of distributed embedded systems requires the development of distributed and multi-thread applications, concurrent multitasking. As already known by software and hardware developers, this leads to a complexity explosion difficult to handle by the human cognitive limitations even if the human is capable of timing multiple overlapping intervals. The number of threads, relationships and temporal interdependencies between them easily exceeds the maximum number of concurrent relationships that adult humans are capable of processing concurrently [HBMB05], thus “a non trivial software written with threads, semaphores and mutexes is incomprehensible to humans” [Lee06].

4.4 Time Representation

Time is a fundamental concept for the development of SCES, thus, the precise and consistent time representation is of utmost importance. This section describes time standards, time formats and the time cycle concept, a cyclic timing representation that suits control algorithms.

4.4.1 Time Standards

Different time standards have been defined to measure the time difference between any two events and establish their position relative to some commonly agreed time base origin, called the epoch, of which TAI and UTC are the most most relevant for embedded systems [Kop97]:

- **TAI** (International Atomic Time) is a high-precision atomic time standard that provides a chronoscopic time-scale (there are not discontinuities) measured in seconds [wwwb].
- **UTC** (Coordinated Universal Time) is a high-precision atomic time standard that defines time with seconds defined by the TAI and needs to add leap seconds to compensate for discrepancies such as the earth's slowing rotation. Each day contains 86400 seconds (24 hours x 60 minutes x 60 seconds) but occasionally the last minute of the day might have 59-61 seconds leading to a day of 86399-86401 seconds. As of 2009, TAI is 34 seconds ahead of UTC [wwwb].

4.4.2 Time Format

Different time format standards have also been defined to represent and identify time instants, from which the following ones shown in Figure 4.3 and described below are the most relevant for embedded systems:

Network Time Protocol (NTP) is the Internet time format standard used to synchronize system clocks among a set of distributed servers and clients that use UTC [Mil06]. The NTP time is not chronoscopic because it is based on UTC and the occasional insertion of a leap second can disrupt the continuous operation of a time-triggered real-time system. The NTP timestamp is 8 bytes long and it is divided into two fields as shown in Figure 4.3, the timestamp (most significant 4 bytes) represents seconds according to UTC that span 136 years and fraction (less significant 4 bytes) represents second fractions with a resolution of about 232 picoseconds. The initial epoch is 1 January 1900.

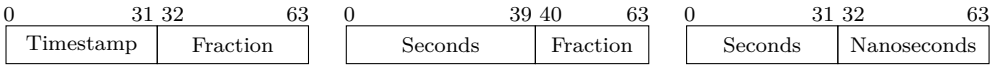
Global Positioning System (GPS) time format is the same as the NTP time format shown in Figure 4.3 except for the initial epoch which is different, 6 January 1980.

Uniform Time Format (UTF) [OMG01] is an OMG time format, used by TTE [KAGS05], represented in binary 64 bits (8 bytes) as shown in Figure 4.3. Full seconds are represented in 40 positive powers of two (up to 30000 years) and fractions of a second are represented as 24 negative powers of two (smallest value is approximately 60 nanoseconds). Based on this format, every instant from the initial epoch, January 1980, to around 30000 years in the future can be represented uniquely with a granularity of about 60 nanoseconds.

IEEE-1588 [iee04] is an international standard issued by the International Electrotechnical Commission (IEC), IEC-61588, that defines a precision clock synchronization protocol for networked measurement and control systems. Time is represented in 64 bits, full seconds are represented as a 32 unsigned integer and nanoseconds as a 32 bit signed integer. The sign of the nanoseconds counting integer, represents a positive / negative timestamp post / prior to the epoch. The epoch depends on the type of grandmaster clock used (e.g., GPS), but for most cases it is based on the PTP epoch, 0 hours on 1 January 1970.

ISO 8601:2004 [iso04] is an international standard issued by the International Organization for Standardization (ISO) for date and time representations: Gregorian dates, time of day, combined date and time of day, and time intervals. For example, $2008-10-23T12:48Z$ defines the time instant 23 October 2008 at 12 hours, 48 minutes *Coordinated Universal Time* (UTC) time, while $P3Y6M4DT12H30M5S$ defines a period of 3 years, 6 months, 4 days, 12 hours, 30 minutes and 5 seconds. This time format is usually used by humans operating with systems, HMI, and by system designers for the time specification of system activities.

Natural language standardized expressions such as the one defined by the VHDL standard [IEE94], time value and time unit (e.g., 10 nanosecond). This time format is powerful, natural, simple and widely used to express time.



(a) NTP **(b) UTF** **(c) IEEE-1588**

Figure 4.3: NTP, UTF and IEEE-1588 time formats (64 bits, 8 bytes).

4.4.3 Time Cycle

The temporal structure of a typical distributed real-time control system is cyclic, where each control cycle can be decomposed into a number of steps [OESHK07, OESHK08]. The timing descriptor of each step (T_{step}), as shown in Equation 4.1, is composed by the period (p) and the phase (ϕ) relative to the cycle start. This cyclic representation suits real-time control system timing representation better than linear model, thus reducing the cognitive complexity in the design of such systems.

$$T_{step} =< p, \phi > \tag{4.1}$$

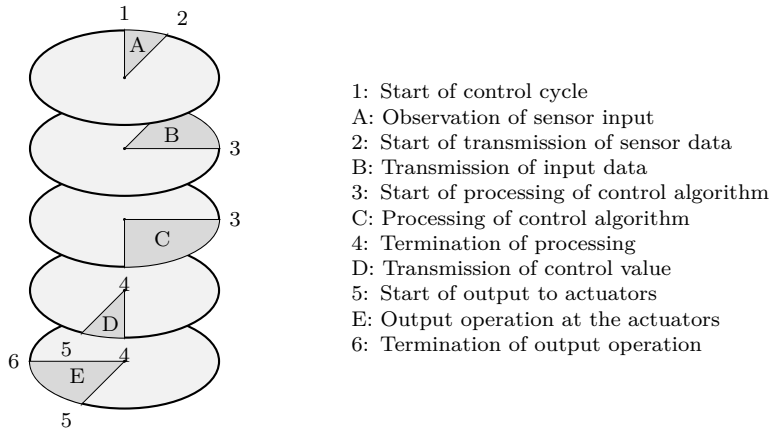


Figure 4.4: Cyclic period with temporal alignment in control loops [OESHK08].

4.4.4 Temporal relations algebra

The study of temporal representation and reasoning is a core area of research in different domains such as Artificial Intelligence (AI) [PSPG00]. This section defines the basic algebra to express time instants, time intervals and time interval relationships to be used within the current document.

Time instant

As previously explained in Section 4.2.1, a time instant is a cut in the timeline. All possible basic temporal relationships that might be established between any two time instants (t_a, t_b) are described below:

- $t_a = t_b$: Both instants are equal, simultaneous.

- $t_a \neg t_b$: Both instants are different, exclusive.
- $t_a < t_b$: t_a precedes t_b.
- $t_a \leq t_b$: t_a precedes or equals t_b.
- $t_a > t_b$: t_a follows t_b.
- $t_a \geq t_b$: t_a follows or equals t_b.

Time interval

Time intervals (e.g., X) can be represented by modeling the endpoint instants, beginning instant (X^-) and ending instant (X^+) [Bru72]. According to this definition a zero duration interval, instantaneous, might be defined as $X^- = X^+$.

Time interval relationships

The following Table 4.1 shows all possible basic temporal relationships between ordered pair of time intervals [All86].

Relation	**Symbol**	**Inverse Symbol**	**Diagram**	**Relations on Endpoints**
X before Y	$<$	$>$	X: ——— Y: ———	$X^+ < Y_-$
X equal Y	$=$	$\neq$	X: ——— Y: ———	$(X^- = Y^-) \wedge (X^+ = Y^+)$
X meets Y	m	mi	X: ——— Y: ———	$X^+ = Y^-$
X overlaps Y	o	oi	X: ——— Y: ———	$(X^- < Y^-) \wedge (X^+ > Y^-)$ $\wedge (X^+ < Y^+)$
X during Y	d	di	X: —— Y: ———	$((X^- \geq Y^-) \wedge (X^+ \leq Y^+))$
X starts Y	s	si	X: —— Y: ———	$X^- = Y^-$
X finishes Y	f	fi	X: —— Y: ———	$X^+ = Y^+$

Table 4.1: Temporal relationships between ordered pair of time intervals [All86].

4.5 Time Abstraction Models

As previously explained, instantaneity and simultaneity could be a source of indeterminism if not properly handled. In order to tackle this issue different time abstraction models shown in Figure 4.5 and additional mechanisms could be used in the development of RTES.

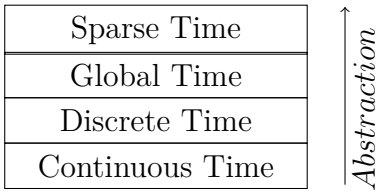


Figure 4.5: Physical time abstraction models.

Continuous / Dense Time: In a dense time model time has a continuous nature as shown in Figure 4.6(a) and Equation 4.2. Time advances continuously, time instants (t) are modeled by real numbers and the delay between two events can be arbitrarily small. This time abstraction suits for example open-world physical systems, analog electronic systems and models (e.g., Simulink).

$$\left(\forall t_i, t_k \in \Re^+\right) \wedge (t_i \neq t_k) \Rightarrow \exists t_j \rightarrow t_i < t_j < t_k \tag{4.2}$$

Discrete Time: In a discrete time model, time has a discrete nature where time is advanced by discrete steps (time intervals) and time instants are modeled by positive integers as shown in Figure 4.6(b) and Equation 4.3. Events can only happen at discrete time values (integer value) and the delay between two events can only be a multiple of this time step. The discrete time ticks are usually generated by a clock (z) that generates the periodic event called the microtick. This time abstraction suits synchronous systems, synchronous languages (e.g., Lustre), Hardware Description Languages (HDL) and centralized / distributed embedded systems in general.

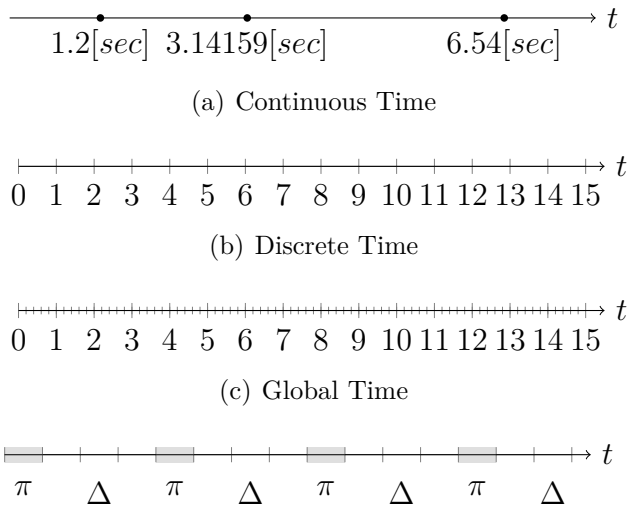
$$\left(\forall t_i, t_k \in N^+\right); (t_i \neq t_k) \Rightarrow (t_i < t_k) \vee (t_i > t_k) \tag{4.3}$$

Global Time: The global time is a common notion of time that helps to establish a consistent temporal order of events based on their timestamps and a time consistent execution of control algorithms, within a distributed RTES. Global time is approximated by the generation of a global time macrotick based on a set of distributed node clocks microticks, as shown in Figure 4.6(c) and Equation 4.4, using a (fault tolerant) clock synchronization algorithm [Kop08a].

$$\begin{gathered} \forall gt, t \in N^{+} \\ gt = clksync(\{t\} \end{gathered} \tag{4.4}$$

Sparse Time: The sparse-time model provides a deterministic consistent distributed system wide notion of simultaneity based on the global time. As shown in Figure 4.6(d) and Equation 4.5, this is done by restricting the occurrence of events that are in the sphere of control of the computer system to the activity intervals of a sparse-time base. The continuous real-time is partitioned into a sequence of alternating intervals of activity of duration π and silence of duration Δ [Kop08a]. This time abstraction suits distributed real-time embedded systems where a deterministic and system wide temporal order of events and notion of simultaneity is required.

$$\begin{gathered} \forall t_i \in \Re^{+}; \forall \Pi_i, \Delta_i \in N^{+} \Rightarrow (t_i \in \{\Pi\}) \vee (t_i \in \{\Delta\}) \\ \forall e\,(t_i) \Rightarrow t_i \in \{\Pi\} \end{gathered} \tag{4.5}$$



(a) Continuous Time

(b) Discrete Time

(c) Global Time

(d) Sparse time

Figure 4.6: Time abstractions.

In addition to this, the delta-delay [Gho99] concept is used to overcome the indeterminism of instantaneous actions executed simultaneously and which might generate race conditions (e.g., logical circuits with loops). The delta-delay (δ) represents an infinitesimal advance in time, which provides a mean for ordering events that are causal but appear to be simultaneous in the time domain. It must be stated that when the delta-delay is used with zero execution time (e.g., VHDL) instead of infinitesimal, this could lead to non deterministic behaviour under certain circumstances [Gho99].

4.6 Time Constraints

A constraint is defined as "a limitation or restriction" [Oxf07] and the most stringent temporal constraints for real-time embedded systems have their origin in the requirements of the system to be controlled (e.g., the plant) [Kop97]. The time constraints covered within this section might apply to jobs implemented as hardware and software, where constraint equivalences might be found in the respective development environments and languages (e.g., VHDL).

4.6.1 Time constraints for data

The temporal validity / accuracy interval of data within a real-time embedded system is limited, or at least for the real-time data. This temporal validity / accuracy interval depends on the nature or source of the data. For example:

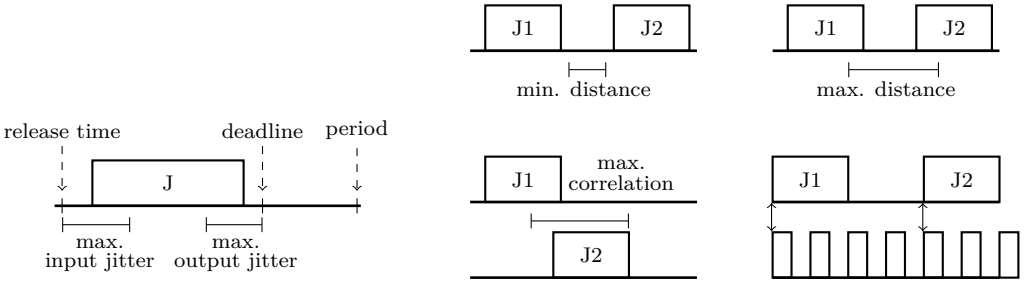
- In system control applications, a *real-time entity* is a significant state variable (e.g., speed) of the controlled plant which changes over time. A real-time image corresponds to the observation of a real-time entity, and a given real-time image is only accurate for a limited time interval which depends on the dynamics of the controlled plant [Kop97]. Therefore, the real-time image accuracy interval is limited and must be taken into consideration as a time constraint.
- Real-Time databases contain persistent and time-varying data, so real-time database transactions have timing constraints [Sno95]. The temporal validity of a given piece of information is given by the deadline associated with this information.
- Multimedia systems process a variety of multimedia information with associated temporal characteristics and temporal validity [WR94].

4.6.2 Time constraints for jobs

Real-time systems timing constraints for jobs have been analyzed and described in the literature [XP93, EJ00, IF00, Xu03, But04], usually assigned to software tasks. A job is considered the smallest self-contained execution entity with a given functional purpose (as in DECOS), which might be implemented as software, hardware or any combination of both. Job timing constraints might be defined as the composition of job and inter-job timing constraints.

From the activation event point of view two types of jobs shall be differentiated [XP93]. Asynchronous jobs are activated by an internal or external event

(event-triggered) while periodic jobs are activated by a periodic event (time-triggered), thus executed repeatedly once every fixed period of time. The k^{th} of a periodic job J_p is denoted by J_p^k.



(a) Job (b) Inter-Job

Figure 4.7: Job and inter-job timing constraints [EJ00].

4.6.3 Job timing constraints

Job timing constraints define the time limits with which the job must operate and can be defined as the vector of parameters described in Table 4.2 and Figure 4.7(a) [XP93, EJ00, Xu03]. From all these parameters, Xu identified (prd_p, r_p, d_p, c_p) and (min_a, d_a, c_a) as the required timing constraints for periodic (p) and aperiodic (a) timing constraints definition [XP93, Xu03], thus other parameters might be considered as optional.

Inter-job timing constraints express time limits within which jobs should be executed in relation to others that can also be described as a vector of parameters described in Table 4.3 and Figure 4.7(b).

Job Type	**Parameter**	**Description**
Periodic job (p)	Period (prd_p)	Execution period
	Release time (r_p)	The earliest time it can start its computation, also called phase (φ).
	Deadline (d_p)	The time it must finish its computation
	WCET (c_p)	Worst-case execution / computation time
	Relative deadline (d_p^-)	The maximum tolerable time interval between the activation and end of execution.
	Jitter input (j_i)	Maximum difference between release time (r_p) and execution start.
	Jitter output (j_o)	Maximum difference between deadline (d_p) and execution end.

Asynchronous job (a)	Min. Time Req. (min_a)	Minimum time between two consecutive requests
	Deadline (d_a)	Execution deadline
	WCET (c_a)	Worst-case execution / computation time
	Relative deadline (d_a^-)	The maximum tolerable time interval between the activation and end of execution.
	Jitter input (j_i)	Maximum difference between release time (r_p) and execution start.
	Jitter output (j_o)	Maximum difference between deadline (d_p) and execution end.

Table 4.2: Job level timing constraints.

Constraint Name	**Description**
Distance $D(X\|Y)$	It constrains the distance (in time) between the production and consumption of an output, by a producer and a consumer job. This is for example the case where two jobs are executed in different communicated nodes, and the distance constraint specifies the delay required (WCCOM) to send the output of a job (Y) to the other job input (X): $D(X\|Y) = WCCOM$
Freshness $F(Y\|X)$	Sometimes called propagation delay, it bounds the time it takes for data to flow through the system because data must be produced sufficiently recently in order to be safely consumed by receivers. The freshness constraint defines the RT image 'temporal accuracy interval'. For example, if a system output (Y) requires a system input (X) with a freshness of $10ms$ it is described as $F(Y\|X) = 10ms$.
Correlation $C(Y\|X)$	It limits the maximum time-skew between several inputs used to produce an output. For example, this is required by voters of a fault tolerant distributed system, where the voter requires all inputs with a maximum reception time-skew to produce the voter output. For example, if a system output (Y) requires two system inputs (X_1,X_2) with a maximum time-skew of $2ms$, this is described as $C(Y\|X_1, X_2) = 2ms$.

Separation $S(Y)$	It constrains the jitter between consecutive values on a single output channel, by specifying the minimum and maximum output value production time. For example, if a system output (Y) is delivered with a minimum rate of $3ms$ and a maximum of $10ms$ this is expressed as $S(Y) = [3, 10]ms$
Harmonicity $H(J_r \vert J_s)$	It specifies the harmonicity of job periods because it is usually desirable for the execution period of the consumer jobs to be exactly divisible by the executing period of the sender job. For example, if the receiver (J_r) has a period p_r and the sender (J_s) has a period p_s exactly divisible by 2, the harmonicity is expressed as : $H(J_r \vert J_s) = p_r \div p_s = 2$
Synchrony $SYNC\,(X)$	It constrains whether a set of jobs must start synchronously / simultaneously.
Precedent $P\,(\{J\}_a, \{J\}_b)$	"A process segment i is said to precede another process segment j if j can only start execution after i has completed its computation" [XP93]. Jobs should execute in a given order expressed by a precedence constraint. The expression $P\,(\{J\}_a, \{J\}_b)$ denotes that the set of jobs $\{J\}_a$ precedes the set of jobs $\{J\}_b$.
Exclusion $E\,(\{J\}_a, \{J\}_b)$	"A process segment i is said to exclude another process segment j if no execution of j can occur between the time that i starts its computation and the time that i completes its computation" [XP93]. Exclusion constraints determine whether a given set of jobs are allowed to be executed concurrently. The expression $E\,(\{J\}_a, \{J\}_b)$ denotes that the set of jobs $\{J\}_a$ excludes the set of jobs $\{J\}_b$. Two exclusion types should be distinguished, $E_{any}\,(\{J\}_a, \{J\}_b)$ denotes that if any job of set $\{J\}_a$ is being executed none from $\{J\}_b$ can be executed while $E_{all}\,(\{J\}_a, \{J\}_b)$ requires that all from $\{J\}_a$ are executing concurrently for this to happen.

Table 4.3: Inter-job level timing constraints.

4.7 Real-Time Embedded System

A real-time embedded system is an embedded system in which the correct operation depends on both the logical results of the computation and the physical instant in which these results are produced [Kop97]. From a SCES point of view, the notion of time is a key concern because every SCES is an RTES with at least one hard deadline. The real-time embedded system must interact with

the environment within the time intervals dictated by the environment and / or the controlled system (e.g., controlled plant) [Kop97]. The interactions between the real-time embedded system and the environment (e.g., sensoring, actuating, waiting for stimuli, etc.) might be periodic, aperiodic, sporadic or a mixture of all.

4.7.1 Time in software and HDL

The temporal behaviour of software has been mainly considered not a fundamental issue, so the notion of time has historically been systematically removed from theories of computation. The major goal of the software process has been to develop a functionally correct implementation of algorithms with little regard to the notion of time [Lee99, MCIJ+02, SM03]. Therefore, current MDA based modeling languages such as UML do not support time with the required rigour for the development of distributed SCES [Kop00b]. Nonetheless, MARTE [omg08] aims to be the UML standard extension to support modeling of real-time embedded system[AMPF07, AMdS07, MPP07] (see Section 6.3).

On the other hand, the temporal behaviour of hardware is usually a critical part of the specification, design and implementation. For this reason the notion of time is a key concern in Hardware Description Languages such as VHDL [IEE94]. In VHDL time constraints and relationships are expressed within designs and implementations using the same hardware description language, where time is the only predefined physical type, defined as a 32 bit integer with an ascending range and femtosecond as the default unit. HDLs such as VHDL are based on the discretized time model, in other words, on an underlying discrete-event model of computation.

4.7.2 Real-Time scheduling

The latest instant at which a result must be produced is called deadline and might be classified as soft, firm or hard depending of the effect of missing the deadline. Missing a soft deadline might worsen the QoS of the given service, missing a firm deadline renders the result not useful for its purpose and missing a hard deadline might result in a catastrophe. Thus a SCES is a real-time embedded system with at least one hard deadline, and the SCES must always provide a guaranteed temporal behaviour under all specified load and fault conditions [Kop97]. Thus, a time predictable scheduling is a key concern in the development of SCESs.

The real-time scheduling problem is concerned with the schedule of jobs so that all timing requirements (e.g., deadlines) are satisfied. A schedule is an

assignment of a set of jobs $J = \{J_1, ...J_n\}$ to a set of processors $P = \{P_1, ...P_m\}$ so that each job is executed until completition. That means, a schedule is a function $\sigma : R^+ \rightarrow N$ such that $\forall t \in R^+$, $\exists t1, t2$ such that $t \in [t1, t2]$ and $\forall t' \in [t1, t2)\ \sigma(t) = \sigma(t')$ [But04].

4.7.3 Control Theory

Control theory is a branch of engineering that deals with the automatic control of systems, in such a way that the controlled system (the plant) behaves in a pre-specified desired manner. This kind of systems are deployed everywhere, from a petrochemical plant to the braking system of a car, and the automatic control system is often implemented as a distributed embedded system. Time underpins control theory, classical and modern, because the output (y) of dynamic systems depends on the inputs (u) and time (t), $y = f(u, t)$. These control algorithms are usually executed periodically and have stringent temporal requirements imposed by the plant (e.g., jitter) [Oga01, Kop97]. Therefore, the execution platforms are real-time embedded systems with stringent temporal constraints. The one-to-one concept mapping between control theory and real-time embedded systems, with a special focus of time, is a key issue.

4.7.4 Worst-Case Execution Time (WCET)

The Worst-Case Execution Time (WCET) [PB00] analysis of software is a key concern in the development of safety-critical systems with high availability, in order to proof that required hard deadlines will be met because the execution time of tasks is bounded and known in advance.

4.8 Analysis

From a Safety-Critical Embedded-Systems (SCES) point of view, the notion of time is a key concern because every SCES is a RTES with at least one hard deadline. Therefore, the notion of time, the representation of time, the description of time constraints and ensuring time properties consistency from requirements to execution is a key concern and challenge for the design of RTES and SCES:

- **Time cognitive**, the mental process by which time is perceived and reasoned by engineers (see Section 4.3), plays a central role in the development of real-time embedded systems. However, the main limitation

for engineers in the development of such systems seems not to be related to the ability to perceive time or multitasking, but to the cognitive limitations required to deal with the emerging complexity of multiple interdependent threads and time relationships representation. Therefore, cognitive complexity needs to be tackled. Segmentation could be used in order to reduce this cognitive complexity due to the concurrent multitasking of multiple interdependent threads. Segmentation refers to the "temporal decomposition of complex behaviour into smaller parts that can be processed sequentially" [Kop08a], thus reducing the amount of information that has to be processed in parallel at a given instant in order to avoid human cognitive limitations.

- **Time representation:** There are multiple time-formats that suit the required consistent representation of time and that could be used to represent time in RTES (see Section 4.4.2), from which the standard IEEE-1588 [iee04] seems to be gaining momentum. In addition to this, the time cycle representation suits real-time control systems timing representation better than linear model, thus reducing the cognitive complexity of time representation.

- **Time Abstraction Model:** There is no single time abstraction model valid for all possible contexts and development abstraction levels, but a set of time abstraction models that could be used to provide time determinism for that given context, structure and implementation technology (see Section 4.5). Of course, it is possible to combine all or some of them. For example, any HDL should at least support a discrete-time model and provide a mechanism to ensure time determinism in case of instantaneous and simultaneous execution (e.g., delta-delay).

- **Time constraints** must usually be met by RTES and SCESs. Therefore, the capability to express time constraints (see Section 4.6) using standardized semantics could reduce cognitive complexity in the development of embedded systems, and could help ensure time properties consistency from requirements to the execution. Thus, real-time embedded system modeling languages should provide support to express time-constraints in order to reduce the cognitive complexity and ensure time consistency throughout the development stages.

- **Simultaneity and instantaneity:** Instantaneity and simultaneity could be a source for indeterminism and unpredictability in the time and value domains for embedded systems if not properly handled . Therefore, if time determinism is required (e.g., SCES), the models should provide the required time abstraction for a given target (e.g., sparse-time for

distributed real-time embedded system) and provide required additional mechanisms (e.g., delta-delay) to ensure time determinism in the case of simultaneous and instantaneous events.

- **Modeling languages** to be used in the development of embedded systems should provide required time abstraction models and consistent mechanisms to specify time properties and time constraints using standardized time representations.
  - **MARTE** modeling language is the UML standard extension to support modeling of embedded systems (see Section 6.3).
  - **SystemC** modeling language (see Section 2.8) could be used for the modeling of complete embedded systems, different structures (centralized vs. distributed) and technology / algorithms (software and hardware). It provides the required time abstract models and mechanisms to ensure determinism in the time domain by handling simultaneity and instantaneity properly.
  - **Formal languages** could be used in the development of SCESs, in order to proof the correctness of a given algorithm. However, a real-time property verified in the model can not always be directly transferred to the realization, because a model is only an approximation of its realization in terms of the issuing time of events (model coverage) [JVG03]. The continuous-time abstraction model used in multiple formal languages (e.g., Uppaal) does not match the time abstraction models used in embedded systems: discrete-time, global time and sparse-time. If the timing abstractions available in formal models could also cover them, the model coverage would increase and enable 'realistic' formal proofs that could be widely used in SCES and RTES developments.

The bottom line for mathematicians is that the architecture has to be right. In all the mathematics that I did, the essential point was to find the right architecture. It's like building a bridge. Once the main lines of the structure are right, then the details miraculously fit. The problem is the overall design.

Freeman Dyson

Chapter 5

The meta-model

This chapter describes the E-TTM meta-model (mmE-TTM) [PNOES10, PPO10], which specifies the rules and constructs according to which an E-TTM model is created, and the E-TTM execution framework (xfE-TTM) that corresponds to the meta-model implementation.

5.1 Introduction

The E-TTM meta-model (mmE-TTM) is based on a strict separation of concerns (partition) between computation and communication, where components communicate among them by means of the exchange of messages across communication channels. Events trigger the execution of components, which can either be time-triggered or event-triggered based on their associated triggering event (e.g., clock-events for time-triggered components). The notion of global time, based on the sparse-time concept, is common for all elements (components and communication channels). The mmE-TTM supports the modeling of systems based on Time-Triggered Architecture (TTA).

The meta-model implementation, a C++ library that extends SystemC with the time-triggered Model of Computation (MoC), enables the codesign and execution of E-TTM models in SystemC. The meta-model implementation provides both the meta-model elements required to develop models and the E-TTM execution framework (xfE-TTM) for developed models.

As shown in Table 5.1, there is a natural mapping between basic architectural concepts (see Section 2.2) and SystemC. Based on this, SystemC could be used to model basic architectural components on which mmE-TTM is based, leading to a natural mapping of higher-level mmE-TTM concepts and SystemC concepts as it will be explained throughout this section.

The PIM of a DAS consists of	A SystemC model consists of
• a set of jobs • that communicate via interfaces containing logical ports, • connected to a virtual communication channel • with a common notion of time	• a set of modules • that communicate via interfaces connected to ports • implemented by communication channels • with a common notion of time

Table 5.1: Natural mapping of architectural concepts to SystemC.

5.2 Elements and relationships

As shown in Figure 5.1 mmE-TTM is based on a strict separation of concerns (partitioning) between computation and communication, while the global notion of time is common for both. Components communicate among them by means of the exchange of messages across ports connected to communication channels that provide interfaces. The items in gray correspond to items provided by the meta-model infrastructure. Thus, the developer is responsible for the definition and connection of components following a two-level design methodology [Kop00b, Kop97]: the development of components (define) and the design of the system architecture that leads to the specification of the *Communication Network Interface* (CNI) in the time and value domain (connection).

- **Time**: The notion of time is common for all elements (global time) and supports time abstraction models described in Section 4.5.
- **Entities**, the smallest functional items used to build a model
  - **Components** (c), as described in Section 5.2.3, provide a desired service to its environment across ports with well-specified interfaces.
  - The **communication channel**, as described in Section 5.2.6, is a communication infrastructure for the exchange of messages between components.
- **Relationships** specify how entities are connected and triggered
  - **Events** trigger the execution of components, events can only be time-triggered (clock event) or event-triggered (any other type of event, sporadic or aperiodic).

– **Interfaces (I/F) and ports**, as described in Section 5.2.5, specify the provided communication functionalities, while the implementation of these functionalities is provided by the communication channel. Interfaces are connected through ports (P) to both components and the communication channel.

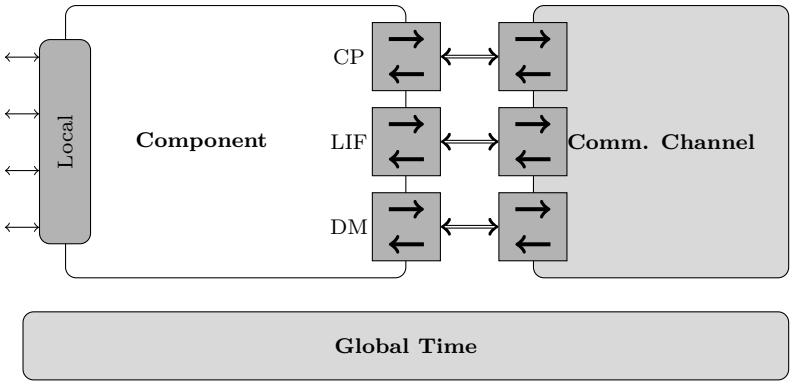


Figure 5.1: mmE-TTM elements and relationships.

5.2.1 Naming

The non-standard usage of the URN (see Section 2.7), just called *Resource Name* (RN), suits the naming needs of mmE-TTM models, because entities could be simply, uniquely and unambiguously identified by their names while abstracted from physical locations. The RN naming scheme is specified as 'rn:NID: NSS' and only alpha-numerical characters can be used.

- *Namespace Identifier* (NID):
  - *job*, *DAS*, *system*: Architectural items described in Section 2.2.
  - *group*: Group of architectural items used for multicast and broadcast communication (see Section 5.2.6).
  - *rte* (real-time entity): State variable of relevance for a given purpose, either located in the environment or in the computer system (e.g., temperature measurement, voltage set-point, etc.) [Kop97].
- *Namespace Specific String* (NSS):
  - Name of the identified item which can be expressed using two different visibility scopes, global or local, as described below.
  - Format (e.g., *double*) and size of the identified item, specified only if required.

As it will be explained in Section 5.4.1, mmE-TTM models follow a hierarchy imposed by the hierarchy of the architectural items. That means jobs are grouped into DASes and DASes into systems. Based on this idea, different visibility scopes can be specified for the NSS name of the identified item, global or local.

- Local: The NSS name corresponds to the name of the architectural item (e.g., *job*).
- Global: The NSS name is described using the local name and all the names of the hierarchical items are separated by dots (e.g., *system.das.job*).

For example, the RN '$rn : job : s.d.j$' globally specifies a job (j) logically located within a system (s) and DAS (d) and the RN '$rn : rte : s.d.rt1 : double$' specifies a real-time entity ($rt1$) of format double and logically located within a system (s) and DAS (d). On the other hand at a given DAS level, a job could locally name another as '$rn : job : j$' without the need to specify the global system and DAS names. And a job could also locally name a local real-time entity as '$rn : rte : rt1 : double$'.

5.2.2 Time

The mmE-TTM is based on a simulation global time, based on the sparse-time concept. The communication infrastructure provides a simulation global time synchronization mechanism so that it is invariant of the selected model execution topology, local when executing the model on a single platform or distributed. Thus, the simulation period / phases are kept constant and the simulation could be executed faster, slower or at the same pace (coupled) as physical time similar to the way in which a video-recorder can be played faster, slower or at a normal pace. Based on this, a distributed model coupled with the physical time will exchange messages with the same content and at the same time instants as the physical RTES should, thus, distributed submodels could be replaced by real RTES subsystems that from the message interchange perspective will not be able to distinguish between submodels and subsystems.

Tick relationships

mmE-TTM models execute components only on each sparse-time macrotick (Γ), as explained in Section 5.2.4. On the other hand, heterogeneous MoC modules described in Section 5.2.3, might require a time tick faster than the

macrotick. The relationship between the global tick (*tick*) and the sparse-time macrotick (Γ) is described in Equation 5.1 where K corresponds to the number of global ticks per macrotick. Figure 5.2.2 describes an example for $K = 5$, where each sparse-time macrotick is divided into five ticks for the integration of heterogeneous MoC modules.

$$\begin{aligned} \Gamma, tick, K \in N^{+} \\ tick = K \cdot \Gamma \end{aligned} \tag{5.1}$$

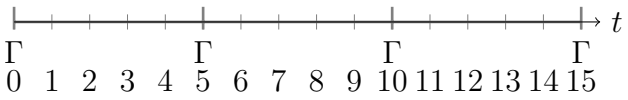


Figure 5.2: mmE-TTM time ticks relationship for $K = 5$.

Time expressiveness

mmE-TTM supports time expressiveness such as description of time properties, values and constraints:

- Time formats: As described in Section 4.4.2, different time formats are used in the embedded domain to manipulate and describe timing values. For each representative time format an associated time format class definition is provided in order to express, manipulate and convert it into other formats (e.g., $ttm\_dt\_time\_format\_omg64$ for UTF). Supported time formats are: *Network Time Protocol* (NTP) [Mil06], *Global Positioning System* (GPS), *Uniform Time Format* (UTF) [OMG01], IEEE-1588 [iee04] and natural language standardized expressions [IEE94].

- Periodic clock: The time-triggered execution of jobs is configured with the timing configuration given by the designer (*time_pdescr_item*) that includes the initial delay, execution period and relative phase. One clock is assigned per job, so that the timing configuration of each job is independent but still based on the same simulation time.

- Time representation: As described in Section 4.4.3, time-cycle representation suits the timing specification requirements of control application. It is possible to express the timing configuration of a control DAS using a time-cycle class instance (*tmm_time_cycle*) where the timing configuration of each job is given by a time-cycle entry (*time_cycle_item*).

- Time Constraints: As described in Section 4.6, job timing constraints specify the time limits with which the job must operate and inter-job timing constraints express time limits within which jobs should execute in relation to others. Job timing constraints definition are already covered in previous descriptions regarding periodic clocks and time representation. Inter-job timing constraints are supported by the class *ttm_time_constraints* that enables inter-job timing constraints to be specified [EJ00]: distance (D), freshness (F), correlation (C), separation (S), harmonicity (H), synchrony (S), precedent (P) and exclusion (E).

SystemC

As explained in Section 2.8.1, SystemC provides a global simulation time of configurable granularity and time deterministic execution of modules as long as a set of recommendations are met. However, SystemC with SystemC-AMS [wa] supports all time abstraction models described in Section 4.5 except for the sparse-time concept.

- Continuous / dense time abstraction model is provided by the SystemC-AMS extension [VGE03b, VGE04].
- Discretized time abstraction model is provided by SystemC natively.
- Global time abstraction model of configurable granularity (macrotick) is provided by SystemC natively, because the simulation time of configurable granularity is common for all the simulation items. Time deterministic models can be designed if appropriate model design restrictions are applied (see Section 2.8.1).

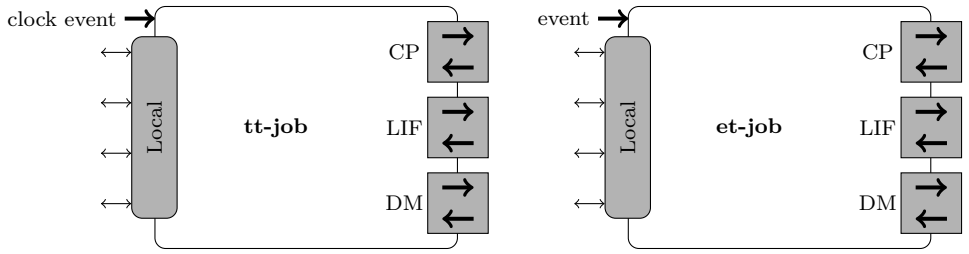
All in all, the SystemC notion of time, clocks and time abstraction models suit all mmE-TTM needs and requirements except for the sparse-time concept. The sparse-time concept is used to restrict the execution and communication of components to the intervals of activity, in order to ensure time determinism even in the presence of simultaneous events. The xfE-TTM extends SystemC with the sparse-time concept, by restricting the occurrence of mmE-TTM clocks events and generic events to the activity interval, the macrotick. Thus, all events that can trigger the execution of mmE-TTM components are restricted to the activity interval.

5.2.3 Components

Components are entities that provide a desired service (e.g., computation) to its environment across a well-specified interface that should maintain its encap-

sulation (value and temporal). Components interact with other components and the outside world by means of the exchange of messages across interfaces connected via ports (LIF, CP, DM and local interface). Time-triggered-components and event-triggered-components shall be differentiated as shown in Equation 5.2, Figure 5.3 and Listing 5.2.3, based on the type of triggering event, time-triggered event (*sc_in_clk clock_event*) and event-triggered event (*ttm_event event*) respectively.

$$\begin{aligned} \forall c \rightarrow c \in \mathbb{C}_{tt} \vee c \in \mathbb{C}_{et} \\ \forall c, \exists! e(c) \rightarrow e(c_{tt}) \in \mathbb{E}_{tt} \vee e(c_{et}) \in \mathbb{E}_{et} \end{aligned} \tag{5.2}$$



(a) time-triggered component (b) event-triggered component

Figure 5.3: Time-triggered and event-triggered job components.

```
SC_MODULE(ettm_ttjob) //TT Job
{
  /* Interface */
  sc_port<ttm_cc_if> lif;
  sc_port<ttm_cc_if>  dm;
  sc_port<ttm_cc_if>  cp;

  /* Events */
  sc_in_clk clock_event;

  /* Constructor */
  SC_HAS_PROCESS(ttm_ttjob);
  ttm_ttjob(sc_module_name name);

  /* Threads:           */
  /*   - clocked thread */
  /*   - thread         */
  /*   - method         */
  virtual void ctask_cthread(){};
  virtual void ctask_thread() {};
  virtual void stask_method() {};
};
```

```
SC_MODULE(ettm_etjob) //ET Job
{
  /* Interface */
  sc_port<ttm_cc_if> lif;
  sc_port<ttm_cc_if>  dm;
  sc_port<ttm_cc_if>  cp;

  /* Events */
  ttm_event event;

  /* Constructor */
  SC_HAS_PROCESS(ttm_etjob);
  ttm_etjob(sc_module_name name);

  /* Threads:           */
  /*                    */
  /*   - thread         */
  /*   - method         */

  virtual void ctask_thread(){};
  virtual void stask_method(){};
};
```

Figure 5.4: SystemC modules for time-triggered & event-triggered components.

Encapsulation

The service functionality provided by a given component could be implemented using different algorithms, heterogeneous MoC [EZ07, HV06, YJL07], different abstraction and refinement levels, different technologies (e.g., software vs. hardware), etc. as long as they do not violate the TT MoC of the component itself. However, this is encapsulated within the component so that the system integrator and other components are abstracted from implementation details. Therefore, the component communication syntax, timing and semantics should be consistent so that which implementation choice has been selected should not be discernible [Kop06].

Integration with heterogeneous MoC

Components internal functionality could be implemented using different MoC as long as the model assumptions described in Section 5.5 are met. Instead of transforming models (e.g., Simulink to SCADE [CCM+03]), the model autogenerated C / C++ code is integrated within components. In addition to this, components could interact with external modules using local interfaces, i.e., open-components. Open components could be a source of non-determinism, and it is the responsibility of the designer to ensure the time and value determinism of the integrated heterogeneous MoC model. Based on this, the following example heterogeneous MoC could be integrated:

- Discrete Time (DT) and Synchronous Data Flow (SDF) MoC are naturally supported by SystemC.
- Synchronous languages (e.g., Lustre) are based on a perfectly synchronous concurrency model in which processes are able to perform computations and exchange information in zero time. The integration within SystemC could be done by integrating the autogenerated source code (e.g., SCADE) or by translating a subset of the synchronous language to SystemC [BS07].
- Continuous Time (CT) MoC can be expressed naturally in SystemC using SystemC-AMS. Simulink models could be integrated by using the autogenerated C source code. CT models should use fixed time steps multiple of the sparse-time microtick, instead of variable time-step [GG06].

5.2.4 Execution

As shown in Equation 5.3 the execution of components is restricted to the sparse-time macrotick that corresponds to the activity interval Π (instanta-

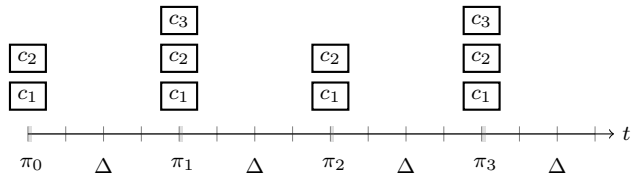
neous), while execution of components should not occur between macroticks (silence interval Δ). As generic events (e) could be triggered at any simulation point in time, the triggering of activation events is delayed till the next macrotick. At a given macrotick multiple components could be executed sequentially by a given simulation platform, i.e., segmentation of concurrent components. Any given sequential order of execution would not violate time determinism if race-conditions among executing components are avoided. Two restrictions should be applied for this purpose: output messages should not be delivered during the execution macrotick to avoid modifying current input messages queue, and generated events should not trigger activation events till next execution macrotick.

$$\begin{aligned} (\Pi, \Delta) &= (0, \Delta\Gamma) \\ e_{tt} \in \{\Gamma\}, e_{et} \in R \rightarrow e(c) \in \{\Gamma\} &\Rightarrow x(c) \in \{\Gamma\} \end{aligned} \tag{5.3}$$

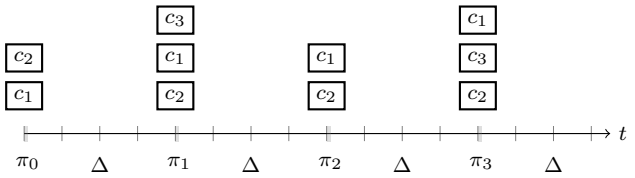
The time domain determinism of components execution is ensured as described below. Based on this, both example executions shown in Figure 5.5 are equivalent and always produce the same outputs at the same sparse-time macrotick.

- Sparse Time: The execution of job components is restricted to the sparse time abstraction model as shown in Figure 5.5 with an activity (π) and silence ratio (Δ) of 0/1. That means that components can only execute at each periodic microtick instant (zero execution time, $\pi = 0$), and not between microticks (silence interval, $\Delta = 1[microtick]$). Therefore, all time-triggered events (clock events) and event-triggered events that trigger the execution of components are restricted to this sparse time.

- Delta-Cycle: If multiple components are triggered simultaneously, all triggered components are executed sequentially and instantaneously (zero time). The order in which these components are executed is not important (time determinism is not affected), because race conditions are not possible, due to the delta-delay message delivery approach provided by the communication infrastructure where messages sent are not delivered until the next delta-cycle (infinitesimal time delay). If any component uses local interfaces, open component, the communication is not restricted by the communication infrastructure and therefore it is the responsibility of the designer to ensure that race conditions and communication 'hidden channels' are not possible.



(a) Sparse-time execution A



(b) Sparse-time execution B

Figure 5.5: Equivalent sparse-time executions.

5.2.5 Interface (I/F) & port

Components communicate with other components via ports, which provide a given interface implemented by the communication infrastructure. The following Listing 5.2.5 describes the LIF, DM and CP interfaces with virtual operations to be implemented by the communication channel. The LIF communication interface supports the sending / receiving of messages to / from other jobs and the access to real-time entities using push / pull mechanism. DM and CP interfaces are based on the LIF interface, because at a PIM level there is not restriction and therefore they support the most complete / stringent interface including the Real-Time Service.

- Send / Receive messages, where the message content is of type 'variant':
  - Send a given message, with content (*msg*) and identifier (*id*) to a job whose name is given by an RN (see Section 5.2.1).
  - Receive next message data-type (*ttm_dt_msg*), which contains the message content (*ttm_dt_msg_content*) and message header (*ttm_dt_msg_header*). If no message is available null message is received.
- Push / Pull real-time entity:
  - Push a given real-time entity image (*rt_image*) value to a given real-time entity whose name is given by an RN (see Section 5.2.1).
  - Pull a given real-time entity image (*rt_image*) value from a given real-time entity whose name is given by an RN (see Section 5.2.1).

```
struct ttm_com_lif : public sc_interface
{
    /* Send / receive messages */
    virtual void send(ttm_dt_rn rn_job,
                      ttm_dt_id msg_id,
                      ttm_dt_variant &msg) = 0;
    virtual ttm_dt_msg receive() = 0;

    /* Push/Pull State messages */
    virtual void push(ttm_dt_rn rn_rte, ttm_dt_variant &rt_image) = 0;
    virtual ttm_dt_variant pull(ttm_dt_rn rn_rte)    = 0;
};
struct ttm_com_dm : public ttm_com_lif {};
struct ttm_com_cp : public ttm_com_lif {};
```

Figure 5.6: LIF, DM and CP interface class definition.

5.2.6 Communication

Components interact with other components and the outside world by means of the exchange of messages across ports, with interfaces provided by the communication channel (communication infrastructure). The message exchanging is quasi-instantaneous (delta delay) and follows the fate-sharing approach where no unnecessary state information is stored. As the functionality of the interfaces is provided by the communication channel, they could be replaced (e.g., TTE) and refined without affecting the computational components.

Time domain determinism

As described in Section 2.5 each component has three independent bidirectional ports for sending and receiving messages (LIF, CP, DM). As shown in Equation 5.4 whenever a component sends a message (during activity interval) the message is delivered to communication infrastructure which generates a unique header information: timestamp, message identifier, job identifier and automatically incremented message counter per port. The communication infrastructure delivers all sent messages to destination component ports during the silence interval.

Each component port has a message queue where all incoming messages are sorted following a deterministic sorting algorithm, based on a string key generated from the header content. The messages sorting criteria selects the lowest unique key value that corresponds to the lowest timestamp (temporal order), message identifier, job identifier and message counter. Thus, segmentation of simultaneous messages is provided by this algorithm because it enables the sequential deterministic sorting of simultaneous messages to be carried out. This algorithm does not impose a design constraint because each TTP provides a

time deterministic sorting algorithm for time-triggered messages that replaces it at system implementation phase.

$$\begin{aligned} \forall c, m &\rightarrow \exists! header \\ header &= [timestamp, m_{id}, job_{id}, m_{counter}] \\ \forall c_a, c_b, m &\rightarrow \begin{cases} c_a.port \rightarrow s\,(m) \wedge t = \Gamma \rightarrow c_b.port \\ c_b.port \leftarrow r\,(m) \wedge t = \Gamma + \delta \leftarrow c_a.port \end{cases} \end{aligned} \tag{5.4}$$

In the real world, computational activities and data exchange require a certain (possibly bounded) amount of time that can be represented by the delayed transmission of output messages by a given number of macroticks. Figure 5.7 shows an example communication between components $c_1 - c_2 - c_3$, where the transmission delay between components $c2 \rightarrow c1$ is set to zero macroticks and the delay between components $c3 \rightarrow c2$ is set to one macrotick.

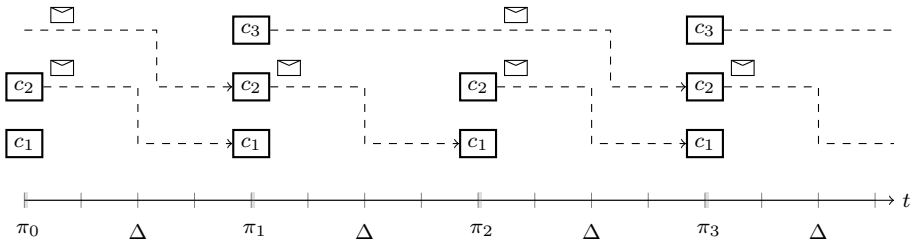


Figure 5.7: Communication with message transmission delay.

Unicast, multicast and broadcast

The default sending of messages from one component to another is unicast, point to point. In order to enable multicast communication, a 'group' RN must be specified and the list of components that integrate this group could be specified statically (compilation time) or dynamically by adding new components to the list during run-time (compilation time permission is required). Whenever a sender component sends a message to a group RN, the message is cloned and forwarded to each receiver component specified in the group list. The group 'rn:group:*' specifies the special and reserved group definition for broadcast, any message sent to this group will be cloned and forwarded to all components if broadcast communication is enabled. The following list describes some example group definitions:

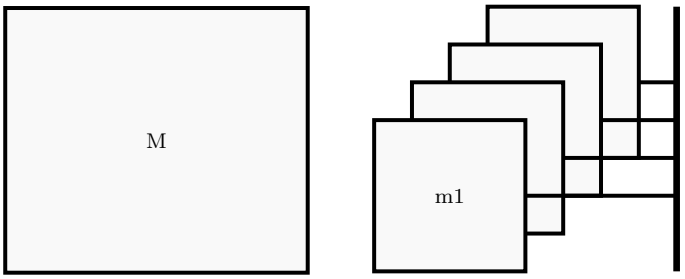
- 'rn:group:g1': This is a local scope group definition of name 'g1', locally visible within the hierarchical level of the sender component.

- 'rn:group:s.d.g2': This is a local group definition of name 'g2'(specified within the system of name 's' and the DAS of name 'd') that can be globally visible within the complete model.

- 'rn:group:*.g3': This is a model global group definition of name 'g3', globally visible within the complete model.

5.3 Simulation topology

As shown in Figure 5.8 mmE-TTM execution supports both centralized and distributed execution topologies, while providing deterministic execution of the overall model in the simulation time and value domain under the model assumptions described in Section 5.5. That means that the same exact results are generated, in the simulation time and valued domain, irrespective of the selected simulation topology (local vs. distributed). This implies that the communication protocol(s) used to connect the distributed models must not only provide the services and properties of the communication infrastructure described in Section 5.2.6, but must also take into consideration communication and synchronization requirements because they must connect two different time islands (simulation time and physical time) in a consistent manner.



(a) Centralized model (b) Distributed model

Figure 5.8: Simulation topology, centralized and distributed model.

5.3.1 Open vs. close simulation

In a close simulation scenario there is no time relationship with the open world physical time and the notion of time is only based on SystemC time. Alternatively, in an open simulation scenario there is a time relationship with the open world physical time (e.g., synchronization). Table 5.2 describes time terms of interest for both scenarios:

Scenario	Term	Description
Close / Open	Simulation time reference	SystemC time, a discretized time abstraction model of configurable granularity.
Open	Physical time reference	Open world time reference based on the Newtonian physics concept of time (wall clock).
Close / Open	Simulation time	Time instant or duration referenced to SystemC time (simulation time reference).
Open	Physical time	Time instant or duration referenced to the physical time reference.
Open	Execution time	Physical time required to execute a simulation, total or partial.
Open	Execution time jitter	Physical time difference between the ideal synchronization instant and the instant when simulation resumes execution after synchronization.
Open	Average ratio	Average relationship between simulation and execution time, $r_{avg} = execution\ time/simulation\ time$. If $r = 1$, execution time and simulation time are equal. If $r < 1$, execution time was smaller than simulation time (accelerated simulation). If $r > 1$, execution time was bigger than simulation time (slower simulation).
Open	Synchronization ratio	Synchronization relationship between simulation and physical time references, $r_{sync} = physical\ time/simulation\ time$. Both simulation and physical time references are coupled.

Table 5.2: Time terms for open and close simulation scenarios.

5.3.2 Communication

Communication, the sending and receiving of information among nodes, could be implemented using a wide variety of communication protocols. In any case, as opposed to the simulation time which does advance until all sent messages have been delivered, physical time advances while messages are being delivered. Therefore, whenever the distributed model is intended to be executed at the same pace as physical time, the developer must ensure that appropriate measures are taken to ensure that all sent messages are delivered within physical time deadlines. If the communication protocol used is a TTP, such as TTE, this could be systematically guaranteed at the design stage.

If the model is executed decoupled from physical time, the time required for

delivery of messages can only affect overall simulation performance. If a TTP is used for this purpose, and the physical time taken by simulation macroticks or periods can be bounded (known WCET), then it is possible to specify a communication period multiple of the simulation period and execute the overall distributed simulation with a given synchronization ratio. The equivalent physical time is somehow extended like a piece of 'chewing-gum' to accommodate simulation speed ratios, while maintaining constant period-phase relationships as shown in the following Figure 5.3.2.

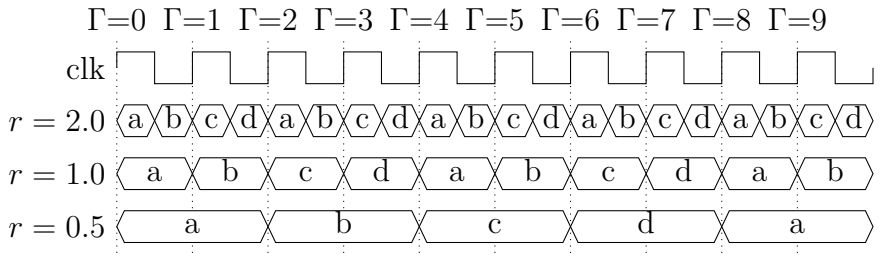


Figure 5.9: Example timing diagram, communication with different time ratios.

5.3.3 Synchronization and simulation global time

In order to ensure simulation time consistency in a distributed topology, a synchronization mechanism that provides a consistent simulation global time is required. For this purpose, the mechanisms could be different for different specific protocols:

- Generic protocols (e.g., Ethernet), could be used in a central master topology where one of the distributed models acts as master and the others as slaves. Whenever each slave reaches the next macrotick, a single message is delivered to the central master that contains the new macrotick value and all messages to be delivered outside the submodel. The central master waits and processes all incoming messages from slaves, one and only per slave submodel, and whenever all messages have been processed a message is sent to each slave confirming the new macrotick and including the messages to be delivered to this specific submodel. This mechanism ensures that all distributed submodels are synchronized and share the same notion of simulation global time.

- Time-Triggered Protocol (e.g., TTE), could be used in a distributed topology in which all submodels are entities that exchange messages among each other, each submodel being a component. The usage of TTP

is a natural approach for the composable distribution of submodels, because the mmE-TTM itself is based on the TTA and the communication infrastructures is based on the same foundation as TTP protocols.

5.4 Complexity management

In order to reduce cognitive complexity, it is preferable to have a modeling language with a small number of well-specified concepts and relationships, that focus on the essential properties (e.g., time and value domain), with a clear structure and model functions, with optional formal notation and clearly stated model assumptions [Kop08a]. So, these recommendations have been followed in the definition of the mmE-TTM in addition to three different strategies that can be used to achieve simplicity [Kop08a]: abstraction, partitioning, segmentation and hierarchy.

5.4.1 Hierarchy

Hierarchy provides simplification by means of system partitioning and abstraction. mmE-TTM models follow the architectural hierarchy described in Section 2.2 where each entity could be represented as a component. Thus, models correspond to the composition of interconnected systems, DASes and job components. As shown in Figure 5.11, a DAS is a composition of interconnected jobs with an additional $i-component$ for gateway purposes so that the internal details and internal communication of the DAS are hidden from outside. Based on this, the DAS itself can be considered to be a component, a hierarchical component (h-component) composed of multiple internal components. The same could be applied to a system, which could be considered to be an h-component composed of multiple internal components (DAS), and so on. Thus, as shown in Figure 5.10 the concept of component is common for all hierarchy levels, from system to jobs, reducing the cognitive complexity of the design.

This approach also enables the encapsulation of the component internal design, IP protection, so that for example a given supplier could design and deliver a DAS as a compiled library. The internals of this component, e.g., job implementation details, are completely hidden and only the information to be exchanged by the $i-component$ is visible. As the RN for jobs and real-time entities can be different inside and outside the component, the external integrator would not even know the number and real names of the internal jobs and real-time entities.

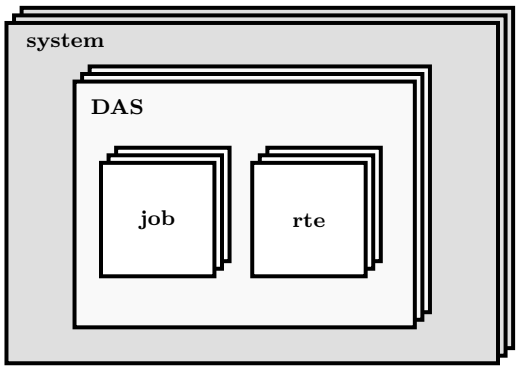
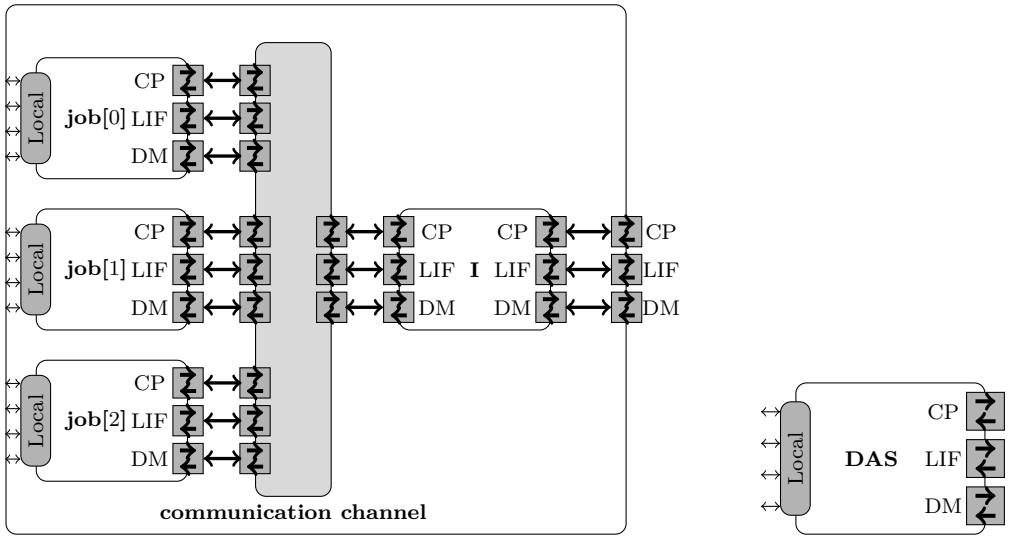


Figure 5.10: mmE-TTM model hierarchy.



(a) DAS composition (b) DAS h-component

Figure 5.11: DAS component composition and representation as h-component.

5.4.2 Abstraction

As explained in Section 2.1.2, abstraction is a simplification strategy to tackle the complexity challenge, in which complexity is reduced by omitting the irrelevant details and focusing on the information relevant for a given purpose. The mmE-TTM approach at PIM level provides abstraction as follows:

- The Object-Oriented Programming (OOP) of the underlying SystemC implementation provides OOP abstraction.
- Encapsulation of the internal functional and timing description of components
- Hierarchy, where the same notion of component abstracts different architectural entities (job, DAS and system).

5.4.3 Partition

As explained in Section 2.1.2, partitioning is a simplification strategy to tackle the complexity challenge, where complexity is reduced by spatial division of the problem into nearly independent parts. The mmE-TTM approach at PIM level provides partitioning as follows:

- Separation of concerns between communication and computation
- Separation of concerns between temporal correctness and data transformation correctness.
- Separation of concerns between application and modeling infrastructure (e.g., communication infrastructure)
- Partition of applications into independent executable entities, components.
- The Object-Oriented Programming (OOP) of the underlying SystemC implementation, provides OOP partitioning of applications.

5.4.4 Segmentation

As explained in Section 2.1.2, segmentation is a simplification strategy to tackle the complexity challenge, where complexity is reduced by temporal decomposition into parts that can be processed sequentially. The mmE-TTM approach at PIM level provides segmentation as follows:

- Sequential deterministic execution of simultaneous components, where at simulation level time determinism is invariant of the (unknown) order in which simultaneous components are sequentially executed.
- Sequential deterministic sorting of simultaneous messages.

5.5 Model assumptions

mmE-TTM PIM modeling approach is stated to be deterministic in the time and value domain under the following model assumptions:

- Components: Integrated components behave deterministically, time and value domain, based on the definition given in Section A.5 - Equation A.7. The developer needs to take appropriate measures when developing the component internal functionality.

- Time domain: The execution framework (xfE-TTM)) provides a notion of common simulation time and the TT MoC provides sparse-time abstraction. The execution platform must provide a consistent notion of physical time.
- Value Domain: The execution platform, compiler options and math library provide bit-exact arithmetic support. This must be ensured by the designer, by using for example a common floating point arithmetic library that ensures bit-identical results among different platforms ([iee85]).
- Scheduling: The xfE-TTM execution framework provides a deterministic scheduling algorithm and describes the restrictions and limitations for the integration of developed modules.
- Communication: The execution of distributed models requires a communication protocol, which provides the same services and properties as the communication infrastructure described in Section 5.2.6 and Section 5.3. This is the case for Time-Triggered Protocols (TTP), such as Time-Triggered Ethernet.
- Fault-Hypothesis: The modeling approach could be used to model fault-tolerant systems, e.g., safety-critical embedded systems. However, the modeling infrastructure does not support the presence of faults, as it has been designed under a zero fault-hypothesis.

5.6 Metrics

The meta-model has been implemented as a C++ library (xfE-TTM) that extends SystemC with the time-triggered MoC. It has been implemented following the requirements described in Section 2.8.1 for the development of time deterministic SystemC models. The main metrics and figures are listed as follows:

- Source code files: 38 source code files, 21 header files (.h) and 17 source files (.cpp)
- Classes and namespaces: 32 clasess and 4 namespaces, where Figure 5.12 shows the hierarchy of most important classes.
- *Lines Of Code* (LOC): 6.351 lines from which 1.096 are executable code lines, 1.360 are declaration code lines, 3.107 are comment lines and 788 are blank lines.

- Complexity metrics: Maximum cyclomatic complexity is 19, average cyclomatic complexity 1,52 and maximum nesting index 6.

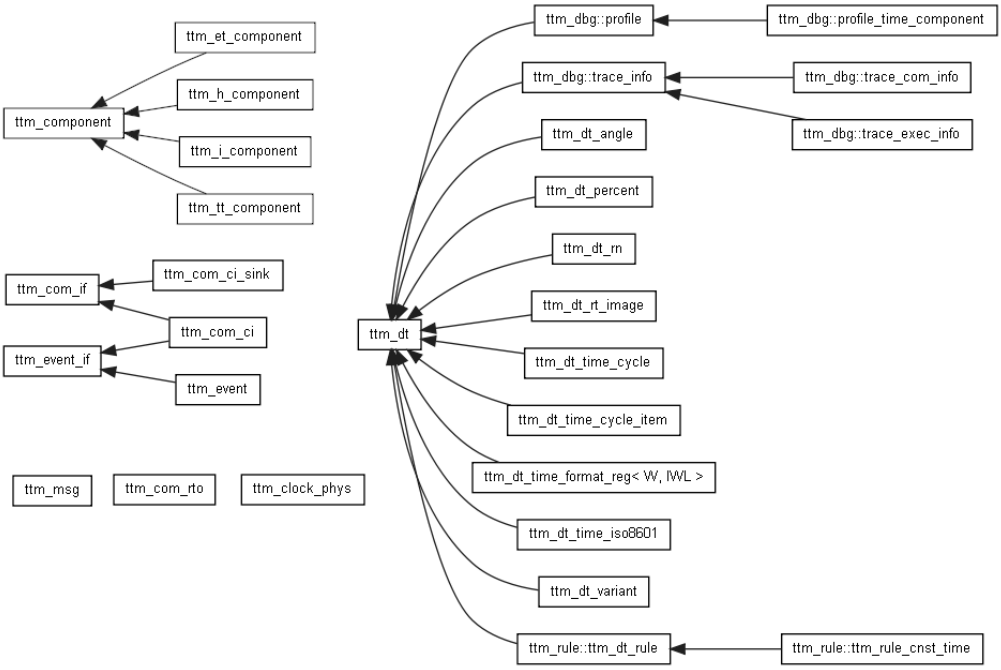


Figure 5.12: mmE-TTM class hierarchy (most important classes).

The ability to ask the right question is more than half the battle of finding the answer

THOMAS J. WATSON

Chapter 6

Related Work

This chapter analyses the modeling of safety-critical and real-time embedded systems, which has been an active research area for decades, identifying points for improvement and ideas that could be used and combined. This chapter is called related work, because multiple modeling approaches could be combined during a development process, and the proposed E-TTM should also be combined and related to others.

6.1 Synchronous Languages

Synchronous languages are a well established and mature programming technology (over 25 years) that enables the modeling, specification, validation and implementation through code-generation of real-time embedded applications.

Description: Synchronous languages are based on a perfectly synchronous concurrency model in which processes are able to perform computations and exchange information in zero time, the reactions to input signals are instantaneous and occur at discrete logical instants called ticks. This simplification is based on the Newtonian physics model (planets evolve in a deterministic and perfectly synchronous way) and provides simplicity, time determinism within synchronous model assumptions and technology-independence at the language level. However, the real implementation and execution is governed by the equivalent of the vibration physics model in which actual response propagation times depend on implementation details [Ber00].

Synchronous languages provide a deterministic behaviour based on the presence of a single global clock and are built on a common mathematical framework that combines [Mar03, BCE+03, Hal98, PBdST04, Ber03, Ber00]:

- Synchrony: Time advances in locksteps with one or more clocks and the program progresses according to successive atomic reactions.
- Deterministic concurrency: They support functional concurrency and rely on notations that express concurrency in a user-friendly manner. The key advantage of using a solid mathematical foundation is the ability to reason formally about the operation of the system along with the availability of formal verification tools. This facilitates certification because it reduces ambiguity and makes it possible to construct proofs about the operation of the system.

The three most mature synchronous languages have been developed by different French universities:

Esterel is a single-rate concurrent imperative synchronous language, based on formal semantics and suited for describing control-dominated reactive systems at the system level. Intuitively, an Esterel program consists of a collection of nested, concurrently running threads described using a traditional imperative language that communicate through signals, which behave like wires in digital logic circuits. From the beginning, optimization, analysis and verification were considered central to the compilation process [BCE+03, Hal98, PBdST04, Ber03, Ber00, LBH06]. The INRIA provides a free version of the Esterel compiler.

Lustre is a single-rate declarative synchronous language based on the data-flow model, in which the usual formalisms are either systems of equations (differential, finite-difference, boolean equations) or data-flow networks (analog diagrams, block-diagrams, gates and flips-flops) . It is single-rate because only flows with equal clocks can be combined by the operators, which ensures that the program can be computed with finite memory [Mar03, HLR92, CCM+03, SSC+04, Ber00]. As an example, the mathematical expression $S_n = 2 \times (X_n \times Y_n)$ is translated into Lustre as follows $S = 0 \rightarrow 2 \times X \times Y$, where the initial value for S is 0. Lustre provides the notion of node to help structure the program. A node is a function of flows that takes a number of input flows, executes a system of equations and defines a number of output flows. In addition to this, it provides clocks and activation clocks functionalities.

Signal is a declarative programming language for real-time applications that defines the data-flow as signals with an associate clock, which is the sequence of instants in which the signal has a value [Mar03]. It is similar to Lustre, but it allows oversampling, and flows with different clocks can be combined by the operators [Ber00].

Analysis: The correct and efficient implementation and execution of synchronous languages is still a challenge due to different reasons such as:

- **Consistency**: Synchronous languages rely on the synchrony model and parallelism to describe discrete-time models that can be formally verified. However, "analysis must typically make assumptions about the execution platform, the external environment, and operator responses, any of which may turn out to be unwarranted" [JTM07]. Therefore, a key challenge is to systematically ensure that the implementation and execution of these programs stills preserves the properties and constraints required by the designed model.
  - **Simplification**: While synchronous languages are based on the simplified synchrony model of computation, implementations and execution environments are governed by the less simple vibration model in which actual response propagation times depend on implementation details [Ber00]. Therefore, a systematic verification of the complete application and execution framework is required to ensure that the synchrony model assumptions are met.
  - **Properties and constraints:** "Determinism should be preserved by the program whenever it is an essential feature of the specification" [Ber03], thus the behavioural determinism (time and value domain) described with the synchronous language should also be preserved by the application and execution framework. It is important to emphasize that "A small timing deviation between the model and its realization may cause properties that are valid in the model to be invalid in the realization" [Hua05].
  - **Parallelism** in sequential processors is simulated by the concurrent execution of threads scheduled by a RTOS [LBH06, Edw99] and executed within an execution framework. Therefore, the complete application and execution framework must be analyzed to ensure that all hard-deadlines are met.
  - **Technology:** The execution framework software (e.g., RTOS and middleware) is commonly defined by programming languages such as C or Ada that have no or weak notion of time. The formal scheduling analysis required to ensure that all hard-deadlines are met might be generally infeasible and multiple restrictions in the execution environment are enforced in order to preserve time predictability (e.g., formally verify the scheduling algorithm, formal bound of WCET of the safety related software, etc.). In the same way, the execution framework hardware and communication protocol also need to be time predictable.

- **Execution framework**: Synchronous programs are generally converted to commonly used languages such as C. However, the concurrency, signals and preemption mechanisms of synchronous languages such as Esterel behave very differently from C [PKjE06]. Therefore, the generated code usually requires glue code and emulates the reactive features of the language, which could also impact performance because "the resultant code is thus inefficient and bulky" [RSD04, TC03]. In addition to this, concurrency is simulated by the concurrent execution of threads and the overhead of context changes can be considerable in large projects [LBH06, Edw99].

- **Distributed target**: "Compiling synchronous languages into code for distributed architectures is obviously a challenge" [Hal98]. Lustre and TTA have already been combined by the product SCADE-TTA, Lustre for the application components and TTA for the communication [DLSMG04]. However, a holistic approach that provides a distributed time predictable execution framework that goes beyond the TTA communication is missing.

6.2 Giotto and TDL

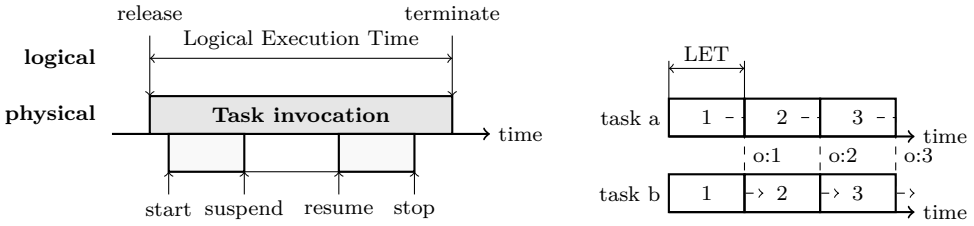
Giotto and TDL are time-triggered languages based on the LET approach developed by the University of California (Berkeley) and designed specifically for embedded control applications programming. They both consider time as a key element to be precisely described and provide an interesting approach towards the development of time predictable real-time embedded systems.

Description: Giotto [HHK01, HK02, KSHP02, Sze05, KSHP02] is based on an abstract programmer's model that provides timing predictability for the implementation of embedded control systems with hard real-time constraints that exhibit time-periodic and multi-modal behaviour as in automotive, aerospace and manufacturing control. It provides a clean separation between the platform independent computation, that can for example be generated from a Simulink model, the I/O timing functionality code generated from the Giotto model (E-code) and the platform dependent concerns of software scheduling and execution. At the design stage, the Giotto compiler must find a suitable schedule that satisfies the jobs execution timing constraints. During run-time a virtual machine, called the E-machine, executes the timing code (E-code) while the functionality code is scheduled by the operating system's scheduler. A Giotto system consists of the following entities:

- Jobs are the basic functional entity that read sensors, perform computations and update output ports and actuators.
- Drivers copy data between ports and physical devices, sensors and actuators, satisfying the synchrony assumption (zero time execution).
- Ports are typed memory locations, for carrying the system state and inter-job communication.

LET (Logical Execution Time) [HHK01, HKS05, PT05], as shown in Figure 6.1, represents the time frame in which a given job must be executed, and it is defined by the 'release' and 'terminate' time instants in which the inputs and outputs are refreshed. The compiler must ensure that even if the job could be preempted, it always finishes the execution before the 'terminate' time instant. In order to provide deterministic execution behaviour of a set of real-time jobs, the LET approach avoids race conditions by introducing a delay for the observable outputs. Based on this, which value is in use at which time instant is always specified, thus avoiding race conditions.

TDL (Timing Definition Language) [Tem05, PT05, FHPS04, Tem07] is a high-level textual notation based on LET and Giotto for defining the timing behaviour of real-time applications [Tem05]. The TDL language seems to be an evolution of Giotto, with extended features and complete development environment integration with Matlab / Simulink.



(a) LET (b) LET unit delay

Figure 6.1: LET programming model time boundaries (a) and unit delay (b).

Analysis: The correct and efficient implementation and execution of applications developed with Giotto / TDL is still a challenge due to different reasons such as:

- **Consistency**: Giotto and TDL rely on the synchrony model as synchronous languages, so ensuring consistency of the designed model properties and constraints in the implementation and execution framework is

also a key challenge as described in 6.1. Nonetheless, the Giotto compiler aims to ensure that the logical semantic (functionality and timing) is preserved [KSHP02, HK02].

- **Execution framework**: The execution of Giotto / TDL applications require a virtual-machine, this way it imposes an important development restriction because the virtual-machine should provide the required fault-tolerance and SIL level for each given execution target. This also involves an overhead through predicate checks, calls of wrapper functions, and the copying of ports. Measurements in an example developed system (e.g., helicopter control system) have shown that this amounts to less than 2% of time and the implementation of the embedded machine accounts for 6KB [KSHP02].
- **Distributed target**: TDL supports the distributed execution of real-time embedded systems [MHF04].
- **Codesign**: The development of embedded systems usually require a codesign approach, where a given functionality could be implemented in software, hardware or any combination of both. The Giotto and TDL approach deal with software only.

6.3 MARTE

Description: SysML [sys06] and UML hardly formalize real-time aspects of embedded systems or at least not with the required rigour [Kop00b, AMdS07]. MARTE (Modeling and Analysis of Real-Time Embedded Systems) [omg08] is the UML standard extension to support modeling of real-time embedded system [AMPF07, AMdS07, MPP07], it provides multiple MDA mechanisms to tackle the complexity challenge (e.g., OOP) and enables the generation of executable models by generating automatic code for SystemC models [MPP07].

In MARTE, time representation can be physical (dense or discretized) or logical (related to user-defined clocks) [AMdS07]. The time structure of a design is built using time bases (TimeBase), a totally ordered set of dense or discretized instants, in which a distributed or multi-thread application consists of multiple time bases (MultipleTimeBase) which are a priori independent. The progress of physical time is measured using the model element clock, which is referenced to a given time base. The Clocked Constraint Specification Language (CCSL) enables semantical and graphical representation using stereotypes of clock constraints such as periodicity, coincidence, strict precedence, independence and exclusion [Kyu08].

Analysis: MARTE provides an interesting foundation for the modeling of safety-critical embedded systems [KAVS08], but this is still a challenge for different reasons such as:

- **Consistency**: Ensuring consistency of the designed model properties and constraints up to the implementation and execution framework is also a key challenge
  - **Formalism**: MARTE as UML is a semi-formal language, and this is why it imposes some limitations on the way functionality and time properties can be expressed and verified at model level and during the implementation process up to the (distributed) execution platform.
  - **Time determinism**: While MARTE supports the notion of time and the generation of executable SystemC models, it does not provide clear restrictions on handling simultaneity. Therefore, the execution of simultaneous components could lead to race conditions, because it does not naturally support time execution determinism.
- **Execution framework**: It does not restrict the execution framework.
- **Distributed execution**: It does not restrict the execution topology, distributed or single.
- **Codesign**: It does not restrict the implementation technology, which could be software, hardware or both.

6.4 TMO

Description: Time-triggered Message-triggered Object (TMO) [MHJGK+00, JT06] targets the design and modeling of high-level distributed real-time embedded systems. It supports the functional and timing behaviour specification in an analyzable form, following different refinement steps. A TMO is built using three different elements: the Object Data Store (ODS) stores real-time and non real-time data, the service method activated by a message from outside and the spontaneous method activated by a clock (time-triggered method).

The TMO model for real-time distributed object-computing supports the specification of temporal constraints and provides execution engines (middleware) for ensuring that these constraints are met at runtime [OHK+05]. Distributed TMO objects may interact via remote calls and spontaneous methods cannot

be disturbed by service methods. A TMO execution engine, middleware, is required for the execution of applications structured as TMO networks. The TMO model has been extended to safety-critical domain [OHK+05] by supporting TMO application on top of the TTA.

Analysis: TMO provides a distributed object-computing support that enables the modeling and implementation of distributed real-time embedded systems. However, the modeling and implementation of safety-critical embedded systems based on TMO has some limitations such as:

- **Consistency**: TMO based systems use a globally referenced time base [OHK+05], a global time. However, as explained in [Kop97] the notion of global time is not sufficient to ensure consistent time-stamps of events in a distributed system, even less in the presence of simultaneous events.
- **Execution framework**: The execution of TMO objects requires a middleware both for the simulation and implementation, this way it imposes an important development restriction because the middleware should provide the required safety requirements for each given execution target.
- **Distributed execution**: TMO supports the distributed execution of applications.
- **Codesign**: The development of embedded systems usually requires a codesign approach, in which a given functionality could be implemented in software, hardware or any combination of both. The TMO approach deals with software only.

6.5 IEC-61499

IEC-61499 [wwwd, Vya07] is an open standard for the design and execution of distributed control and automation embedded systems. It aims to overcome the drawbacks in IEC-61131 [wwwd, PLC08] programming languages that do not fit into the new requirements for distributed and flexible automation systems well but which have been extremely successful in industrial automation systems because they are based on the way and patterns of thinking of control engineers [Vya07].

Description: IEC-61499 applications are designed as event-driven block diagrams composed of different types of function blocks: basic, composite, communication and service interface (a function block that interacts with hardware

resources, e.g., inputs / outputs). The IEC-61499 function block is an abstraction that describes an interface and functionality, which can be implemented as software and / or hardware, so that this abstract yet executable functionality description is not dependent on a particular implementation. A basic function block, from which composite function blocks are built, is a platform independent abstraction of a software component that can be programmed in any form supported by the implementation platform (IEC-61131, C, etc.) [Vya07].

The designer designs the application as a function block diagram and maps the function blocks to different resources in different devices. The event-driven execution semantic of the whole system designed using IEC-61499 does not depend on the particular number and topology of computational resources and devices. This design could be processed by an IEC-61499 COTS tool that generates a system configuration to be loaded in each target device. The system configuration is an executable description that contains the description of the functionality and description of the system architecture. The system configuration is feasible (i.e., can run) if each device in it supports the function blocks that are mapped on it [Vya07].

The IEC-61499 also provides new opportunities for intelligent automation systems by enabling the intelligent integration of system services using Web Service (WS) technology. In a distributed environment, each integrated component within the distributed system could inform the main supervisor system about the services that can be provided using the semantic web technologies. The system supervisor could use this information to generate intelligent manufacturing schedules and negotiate with different manufacturing execution units [Vya07].

Analysis: The correct and efficient implementation and execution of applications developed with IEC-61499 is still a challenge for the development of safety-critical embedded systems for different reasons such as:

- **Value domain composability**: Function blocks can be implemented using different programming languages supported by the execution platform, thus, IEC-61499 does not preserve value domain composability. As explained in Section A.3, page 126, the same function block implemented with different programming languages in a different platform could provide non-identical results (not bit-exact).
- **Time domain composability**: The IEC-61499 does not consider time domain with the required rigour and does not preserve time domain composability (see Section A.3, page 126, for further details). The designer can specify execution periods for different sub-applications that need to

be tested in the execution platform because execution time is just a consequence of the mapping of function blocks to different devices. The system configuration feasibility test does not consider time and time-sequence diagrams used to specify service primitives are just a "qualitative specification form as it does not specify exact timing requirements to the services" [Vya07].

- **Predictability**: Due to the IEC-61499 event-driven nature it is difficult to predict the execution order and timing properties of a given application. For example, several function blocks can be activated simultaneously and scheduled by the resource to be executed simultaneously, but this could lead to race conditions and lack of execution execution order predictability which is required for the design of safety-critical embedded systems. This also makes an integrated-architectural approach impossible.

- **Resource and constraints allocation**: IEC-61499 does not address the global resource and constraints allocation problem (e.g., computation time, communication bandwidth, shared components, etc.).

Therefore, IEC-61499 provides multiple interesting concepts for the development of control application in distributed embedded systems (e.g., a function block is an abstract yet executable functionality description, not dependent on a particular implementation or execution platform) but does not seem to be suitable for the development of safety-critical embedded systems due to its limited predictability and composability in the time and value domain.

6.6 AUTOSAR

Description: AUTomotive Open System Architecture (AUTOSAR) [wwwa] is a development partnership among major automotive industry players, formally launched in 2003, for the specification of an open standard for automotive E/E architecture. The industry objective of the AUTOSAR consortium is to "cooperate on standards and compete on implementation" [Sou06], avoiding wasting time and effort adapting existing black-box components to different environments and products. AUTOSAR has nine primary objectives that are listed below [AUT05]:

1. Consideration of availability and safety requirements

2. Redundancy activation

3. Scalability to vehicle and platform variants
4. Implementation and standardization of basic system functions
5. Transferability of functions throughout network
6. Integration of functional modules from multiple suppliers
7. Maintainability throughout the whole "Product Life Cycle"
8. Increased use of "Commercial off the shelf hardware"
9. Software updates and upgrades over vehicle lifetime

AUTOSAR mainly concentrates on the software architecture and aims to improve complexity management of integrated E/E architectures through increased reuse and exchangeability of software modules among manufacturers, suppliers and platforms. It specifies the Virtual Functional Bus (VFB) that provides standardized communication mechanisms and services to the software components, by decoupling the application from the underlying infrastructure [AUT05]: Run Time Environment (RTE) and Basic Software. As the Unified Modelling Language (UML) has become de-facto standard for modeling OO embedded systems, AUTOSAR uses similar concepts to UML 2.0 (e.g., components, ports and interfaces) and goes beyond in the definition of other concepts (e.g., interfaces) [Kor06]. The AUTOSAR methodology supports a distributed, function-driven development process and standardizes the software architecture for each Electronic Control Unit (ECU).

An AUTOSAR software component is a unit of execution described by its ports, which interact only by well defined interfaces according to contractual guarantees, and that in theory can be independently deployed and composed without modification [SG07]. The behaviour of each component is represented by a set of runnables, event-triggered and time-triggered procedures, executed by a set of threads in a task and resource model [FDNG+09].

Analysis: AUTOSAR provides an interesting methodology for the development of software components for the automotive domain industry, in constant evolution and improvement. However the modeling and development of safety-critical embedded systems based on AUTOSAR does not seem to be feasible nowadays:

- Consistency: AUTOSAR lacks a timing model and timing description support which leads to components execution timing inconsistencies and issues that needs to be tackled during the system integration phase

[FDNG+09, Ric06, Ham07]. Nonetheless, the Timing Model (TIMMO) project [wwwh, FDNG+09] aims to provide AUTOSAR with a timing model and a standardized infrastructure for the handling of time specifications.

- Dependability: As stated in the first item of the AUTOSAR objective list dependability is just considered, but not supported.
- Execution framework: The execution of AUTOSAR applications require the availability of a Run Time Environment, a middleware which provides the required execution framework. This is indeed an important restriction for the development of safety-critical embedded systems, because the RTE should be an order of magnitude safer than the application and this is indeed very difficult to achieve even more so when taking the previously described lack of timing model and dependability consideration into account.
- Distributed Execution: AUTOSAR supports the distributed execution of applications. However, due to previously described time inconsistencies and dependability considerations, it does not support the systematic development of distributed real-time and safety-critical embedded systems.
- Codesign: AUTOSAR handles software components only.
- Simulation: AUTOSAR methodology does not consider simulation, partially because the lack of a timing model makes it unclear how distributed real-time applications made of multiple components might execute. This is a considerable drawback, because early simulation helps to evaluate the system in the early phases. Different solutions have been proposed, such as using SystemC during the design process of AUTOSAR-conform systems [KBH+07] and the use of Simulink [aut06].
- Complexity management: AUTOSAR provides several mechanisms to tackle the ever increasing complexity challenge [HSF+04], such as abstraction and partition. However, due to previously specified time inconsistencies it does not systematically support segmentation.

6.7 Periodic Fine State Machine (PFSM)

Description: The Finite State Machine (FSM) model is timeless, so it is not possible to model system properties that are dependent on the progression of real-time such as duration of computations and the limited temporal

validity of real-time data. The Periodic Finite State Machine (PFSM) incorporates the notion of time and overcomes previous limitations [KESHO07]. PFSMs [KESHO07, OESHK07] provide a consistent notion of time that incorporates the notion of global time, sparse time, periodic clock constraints, time-triggered activities and state variables. This enables a concise representation of distributed control systems and reduces the gap between a modeled system and its implementation.

Analysis: PFSM provides an interesting approach for the design and development of distributed state-machines for safety-critical embedded systems:

- **Consistency**: PFSM provides time determinism based on the notion of sparse-time and supports formal analysis through model checking [OESHK07].
- **Execution framework**: It does not restrict the execution framework, but in order to ensure time domain formalism the (distributed) execution framework should provide a notion of sparse-time (e.g., TTA).
- **Distributed Execution**: It supports the development of distributed real-time embedded systems [OESHK07].
- **Codesign**: It does not restrict the implementation technology, which could be software, hardware or both [OESHK07].

6.8 Analysis

As explained in Section 1.1, tackling the complexity challenge, providing a consistent notion of time and preservation of properties through the development process are key challenges for the development of safety-critical embedded systems. State-of-the-art models and methods described in this section provide different solutions of interest for the design of distributed safety-critical and real-time embedded systems, in which multiple models and methods could be used during the development process. However, as discussed in this section, there are several limitations and issues that need to be addressed in the discipline of safety-critical embedded systems design and development [Kop07, HS07, JTM07, KOPS04, BFLS01]. As has been discussed, the E-TTM approach provides multiple benefits, addresses some of these limitations and could be used in conjunction with different modeling approaches, at the expense of targeting a specific architecture (TTA) and meeting a set of restrictive constraints

6.8.1 Preservation of time properties

A key challenge is to systematically ensure that the development and execution of a designed model still preserves the properties and constraints of interest for the system, e.g., time. As explained in Sections 6.1 and 6.3, time properties and constraints expressed in synchronous languages and MARTE cannot systematically be preserved throughout the development process, so consistency problems can arise during the development process. On the other hand, Giotto, TDL, TMO and AUTOSAR rely on a virtual-machine or middleware in order to consistently preserve time properties from the model up to the implementation. But the use of these virtual-machines and middleware for safety-critical embedded systems does not seem to be feasible because they would need to be of an order of magnitude safer than the application with the highest Safety Integrity Level (e.g., SIL-4 according to IEC-61508). This requires an expensive and long certification process (e.g., TTA) and a safety foundation which these virtual-machines and middleware do not seem to have.

The E-TTM provides a consistent notion of time and time properties preservation from the model down to the implementation (if this is based on the TTA), without imposing the requirement of a middleware in the final implementation (see Section 6.8.5). This would constitute a considerable benefit compared with the available solutions, but at the expense of targeting a specific architecture (TTA) and meeting a set of restrictive constraints. Nonetheless, it can also be used to model generic real-time and safety-critical embedded systems by relaxing the imposed constraints, thus the developer is then responsible for ensuring the consistency and preservation of time properties.

6.8.2 Tackling the complexity challenge

All the modeling approaches and methods described in this section, provide powerful simplification strategies and mechanisms, e.g., MARTE provides OOP strategy. However not all of them where created with this purpose as one of the top objectives, e.g., tackling the complexity challenge is not considered in the top nine AUTOSAR objectives [AUT05].

On the other hand, the E-TTM approach as described in Sections 5 and 5.4, has been designed from the foundation with the objective of tackling the complexity challenge by means of simplification strategies such as abstraction, partitioning and segmentation.

6.8.3 Distributed simulation of time-triggered embedded systems

The distributed simulation of embedded systems has been an active research area for decades [HSLGM07]. There is a wide variety of solutions, from generic solutions that integrate heterogeneous applications into a distributed model interconnected by TCP/IP [HA04], to specific solutions that target the distributed simulation of time-triggered embedded systems such as:

- TMO [MHJGK+00, KJ05, JT06], as explained in [HSLGM07] and Section 6.4, represents a successful object-oriented approach that combines time-triggered and event-triggered behaviour for distributed simulation and implementation. However, the underlying notion of time does not support the sparse-time concept derived from the time-triggered MoC and time determinism is not systematically preserved in the presence of simultaneous events. TMO has already been integrated into Time-Triggered Protocols (TTP) at execution platform level [OHK+05].
- TDL supports the distributed execution of developed real-time embedded systems, but no distributed simulation support claim has been found.
- The DECOS tool-chain proposed different solutions for the simulation of distributed time-triggered systems, covering specific solutions such as distributed simulation of time-triggered based systems derived from SCADE [HSLGM07].

The E-TTM provides an additional complementary approach for the modeling and distributed simulation of TTA based safety-critical embedded systems, providing a time deterministic execution framework based on the time-triggered MoC. This constitutes a considerable benefit compared with the available solutions, but at the expense of targeting a specific architecture (TTA) and meeting a set of restrictive constraints. The E-TTM could also be used in conjunction with other modeling approaches previously described by using appropriate communication gateways and restrictions.

6.8.4 Simulation time and physical time decoupling

The E-TTM simulation is invariant of the physical time progression and the distributed simulation topology, this is indeed a powerful property that has not been identified in any other analyzed modeling approach for the development of embedded systems. This property enables the simulation of periodic controls application models (described with time-cycle period and phases) faster, slower

or at the same pace as physical time, but producing the same results at the same simulation time instants always. This is similar to a video-player which can play films faster, slower or at the same pace as physical time.

6.8.5 Execution framework

Whenever possible, the design methodology should not impose restrictions on the final execution framework or those restrictions should provide a clear benefit for the safety-case. For example, TDL requires the availability of a virtual-machine and TMO requires a middleware in the final execution platform, which involves considerable safety limitations (see Section 6.8.1) and limits the applicability of these modeling approaches in the development of safety-critical embedded systems. TMO has already been integrated with Time-Triggered Protocols (TTP) at execution platform level [OHK+05], but it would also need to be certifiable in order to be used in a safety-critical embedded system. On the other hand, synchronous languages, PFSM and MARTE do not impose limitations on the underlying execution framework.

The E-TTM requires the availability of an execution framework (xfE-TTM) for simulation purposes only, which does not have dependability requirements. In order to take full advantage of this modeling approach, the implementation should be based on the TTA that provides a validated and certifiable core technology for the development of safety-critical embedded systems [JSPP04] and a set of certified tools and products already used in the development of safety-critical embedded systems, which provide a clear benefit for the safety-case. Nonetheless, the developer could also choose to implement the TTA based system using custom developments (non-commercial) or could implement the safety-critical embedded systems using a different architectural approach.

6.8.6 Codesign

Whenever possible, the design methodology should enable codesign and not impose restrictions on the implementation technology to avoid technology obsolescence, enable energy consumption trade-off analysis, etc. Previously described Giotto, TDL, TMO and AUTOSAR target the development of software while synchronous languages and MARTE do not impose limitations on the technology to be used for the development of systems. The E-TTM meta-model supports the codesign of systems thanks to the underlying SystemC implementation, which as described in Section 2.8.3 supports the codesign of systems.

6.8.7 Heterogeneity

System designers deal with a wide variety of engineering disciplines (e.g., control engineer, software engineer, hardware engineer, etc.), heterogeneous MoCs, heterogeneous languages, heterogeneous tools, etc. . Therefore, there is a need for heterogeneous specification and modeling support, which is supported to a different extent by each of the previously described modeling approaches.

SystemC has started to play a significant role as a unifying system-level language and the recent standardization as IEEE-1666 is a clear symptom of acceptance and support [HVG+07]. The E-TTM meta-model implementation is based on SystemC, thus as described in Section 5.2.3, it builds on top of SystemC heterogeneity support which must be restricted in order to meet the meta-model constraints. Nonetheless, multiple MoCs could be used to define the functionality of components and integrate at (distributed) model level. Therefore, E-TTM heterogeneity is built on top of an open standardized system-level language as opposed to other modeling approaches.

6.8.8 Conclusion

All in all, the Executable Time-Triggered Model (E-TTM) approach seems to be a valid an interesting approach for the executable modeling and development of safety-critical embedded systems. As previously analysed, it provides multiple potential benefits compared with the available solutions, but at the expense of targeting a specific architecture (TTA) and meeting a set of restrictive constraints. However, the developer could also use the E-TTM to model generic real-time and safety-critical embedded systems by relaxing the imposed constraints, and ensuring the consistency and preservation of time properties. In addition to this, E-TTM could also be used in conjuction with some of the previously described modeling approaches:

- The use of PFSMs for the design of state machines seems to be a natural approach, both are based on the time-triggered MoC.
- A new MARTE profile could be generated for the modeling of TTA based systems, which could be based on the E-TTM meta-model (mmE-TTM) and autocode generation could be based on the E-TTM execution framework (xfE-TTM).
- As explained in Section 5.2.3, synchronous languages could be used within E-TTM by encapsulating the generated code within E-TTM components.

However beautiful the strategy, you should occasionally look at the results

Sir Winston Churchill

Chapter 7

Case Study

This chapter assesses the proposed E-TTM modeling approach with the modeling and analysis of two case study systems: an example industrial real-time control system [PPO10] and a safety-critical odometry system [PAAP10]. Each case study is subject to experimental evaluation, with the execution of simulations using different experimental setups and analysis of collected results.

The mmE-TTM and xfE-TTM described in Section 5 specify the rules and constructs according to which SystemC based time deterministic E-TTM models are constructed if a given set of model assumptions are met. The experimental evaluation performed in this Section aims to validate previously stated time determinism and analyze different time related issues that might arise when the simulation and physical time references are coupled.

On the other hand, the experimental evaluation performed in this Section does not evaluate how supported simplification strategies reduce the cognitive complexity of the design, due to the psychological and subjective nature of the matter. Nonetheless, supported strategies are based on the research of different authors [Kop08a, RE06, JTM07] and strategies proposed by multiple embedded system experts from different domains [OK09].

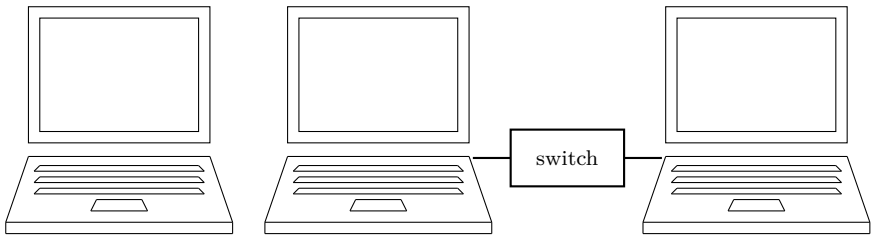
7.1 Experimental setup

The experimental setup defines both the model configuration and simulation platform, and implicitly the experimental results that can be collected. Any given experiment is defined by an experimental setup.

7.1.1 Simulation platforms

Three different simulation platforms shown in Figure 7.1 have been selected for the simulation of centralized and distributed model configurations:

- Single platform, Figure 7.1(a), a laptop with a dual core microprocessor (2.40 GHz) with Windows XP operating system and Visual Studio 2008 development environment (using 'O2' maximize speed optimization).
- Single multicore platform, Figure 7.1(a), the same dual core laptop.
- Distributed platform, Figure 7.1(b), based on the commercial TTE 100 $Mbits/s$ development kit from TTTech [wwwi]:
  - eeePC: An eeePC laptop with Ubuntu 2.6.24-24-rt kernel and GNU g++ 4.4.1 compiler (using '-O3' optimization).
  - Atom PC: An Atom CPU based PC with Ubuntu 2.6.24-24-rt kernel and GNU g++ 4.4.1 compiler (using '-O3' optimization).
  - TTE switch, with support for TTE and Ethernet protocols.



(a) Single / Single multicore (b) Distributed

Figure 7.1: Simulation platforms.

A close communication scenario [Kop08b] is defined for Ethernet based experiments, where no other computer except for the ones defined in the simulation platform are connected to the switch. This is also applied for TTE based experiments, although not strictly required because TTE is a time deterministic communication protocol for time-triggered dataflow (as explained in [KAGS05, Kop08b]) that avoids temporal conflicts in the communication system. TTE based experiments use time-triggered dataflow. Ethernet based experiments are implemented using Winsock with TTE drivers unloaded, and synchronized as described in Section 5.3.3 for Ethernet communication.

7.1.2 Simulation configuration

Each system design under analysis can been modeled and simulated using different configurations such as:

- Plant MoC: The plant to be controlled can be modeled as a Continuous Time (CT) plant model (using SystemC-AMS) or as a Discrete Time (DT) plant model, as described in Sections 5.2.3 and 5.5.
- Centralized vs. distributed: A centralized model uses SystemC based communication channels to intercommunicate components. However, SystemC based communication channels can be replaced by physical communication channels (e.g., TTE) allowing the partition of models into submodels that can be executed in a distributed platform as described in Section 5.3. A single executable file is compiled and linked, containing the whole model and all selectable model partitions. In a centralized execution a single executable instance is executed, while in a distributed execution one executable instance per model partition is executed where the selected partition is an execution parameter.
- Coupled or decoupled: Simulation time (SystemC) and physical time (wall clock) references can be coupled or decoupled as described in Section 5.3.3.

7.1.3 Simulation results

Collected simulation results depend on the selected experimental setup, e.g., with the simulation of a centralized model there is no network protocol analyser data to collect. The following simulations results can be collected (see Section 5.3.1 for a description of time related terms):

- Simulation time trace file(s): Both case studies have a 'trace to file' option that stores component time traces in a log file with the following information: component resource name and component simulation time.
- Execution time trace file(s): Both case studies have a 'trace to file' option that stores component execution time traces in a log file with the following information: component resource name, component simulation time, component execution start and end time referenced with the physical time (computer clock)
- Data trace file(s): When the plant is modeled using CT MoC (using SystemC-AMS) a data trace file stores the value of variables of interest.

- Network protocol analyser data: Wireshark network protocol analyser is used to collect communication messages sent between distributed sub-models, for both Ethernet and TTE communication protocols.

7.2 Industrial real-time control system

The first case study models an example industrial real-time control system [PPO10] where an example triphasic power-electronic evaluation platform [PdAEOSS+08] must be controlled with a Voltage / Frequency (V/F) control, which controls the value and frequency of the voltage applied to an electric load. In addition to this, the control system implements a safety-related function (SIL-2) that protects the power-electronic system from overcurrents. The system is designed using the mmE-TTM and xfE-TTM described in Section 5 and experimentally evaluated with different simulation topologies.

7.2.1 System description

Figure 7.2 shows a simplified deployment model of the complete system. The control system requires the acquisition of the bus voltage, phase voltages and currents, and commands the digital output assigned to the load contactor and six power-semiconductor device drivers. The user interacts with the control subsystem by sending commands and receiving status and monitoring information. Figure 7.3 shows the plant block diagram and Figure 7.4 shows the electric circuit diagram.

- The voltage source provides a constant voltage to the inverter, bus voltage (v_{bus}).
- The inverter block generates a triphasic output voltage of variable value and frequency, as controlled by the real-time control system. It is composed of a sensor (v_{bus}), actuator (do) and six IGCTs ($S\_1, \bar{S\_1}, S\_2, \bar{S\_2}, S\_3, \bar{S\_3}$) with driver commands ($[S1..S]$).
- The load block is a composed of sensors ($i_{u,v,w}, v_{u,v,w}$) and the triphasic resistor and inductance load. Voltage sensors are not represented in Figure 7.4 for simplicity.

The control system executes a control strategy that regulates the on/off switching of the power-semiconductor devices in order to generate a triphasic sinusoidal output voltage of given voltage and frequency set-points (V/F control). The on/off switching of power-semiconductor devices is restricted by a set of rules, e.g, S_x and $\bar{S}_x$ must not be both 'on' at the same time.

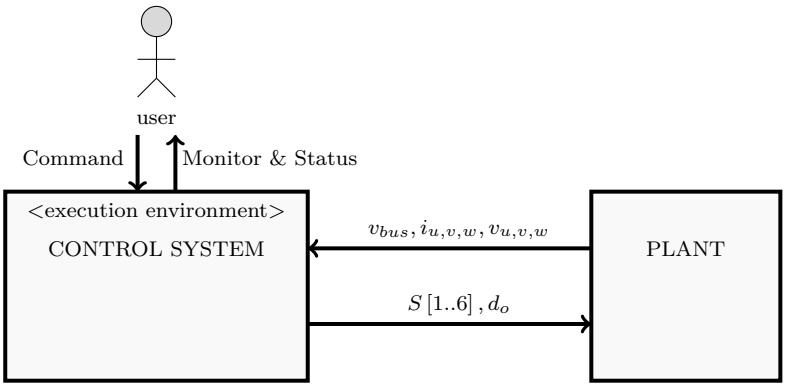


Figure 7.2: System deployment model.

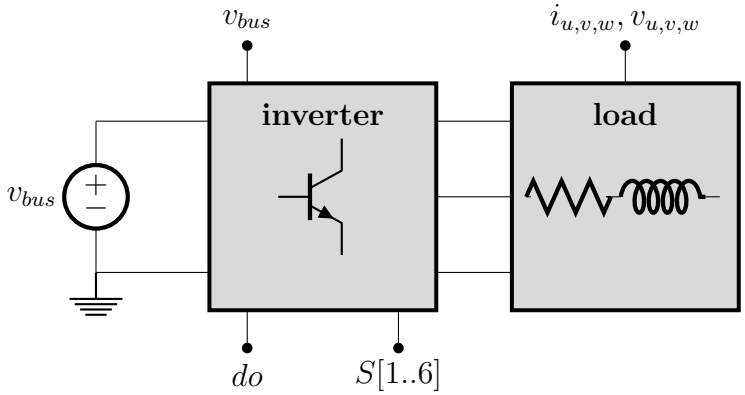


Figure 7.3: Plant block diagram.

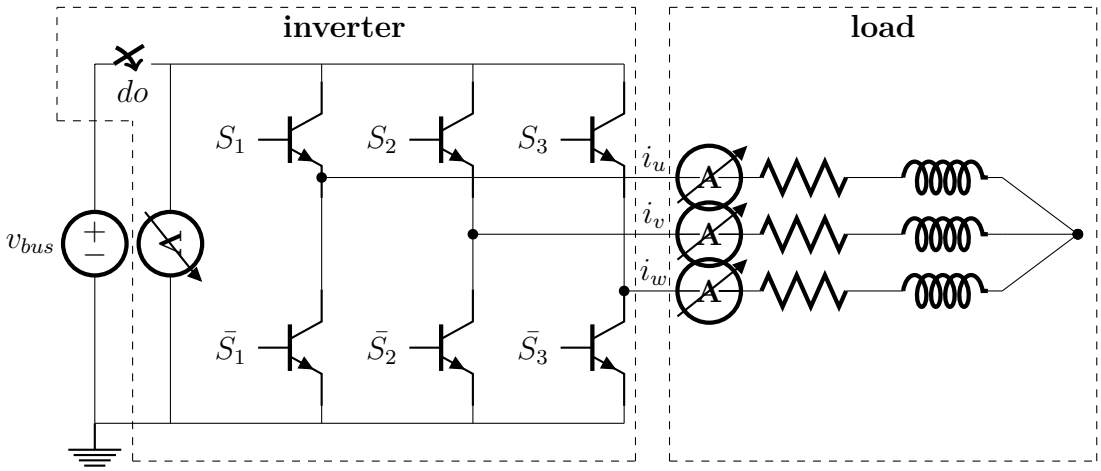


Figure 7.4: Plant electric circuit diagram.

7.2.2 System design

As shown in Figure 7.5(a) the designed system is composed of:

- Components: The User Interface (UI) DAS shown in Figure 7.5(b) has a single job (*job_ui*). The control DAS shown in Figure 7.5(c) is composed of multiple jobs: acquisition (*job_acq*), control (*job_control*), modulation (*job_modulation*) and digital output actuation (*job_act_do*).

- Communication channels: Components are interconnected by communication channels. The system level communication channel that interconnects both DASes can be replaced with a physical communication protocol, e.g., Ethernet.

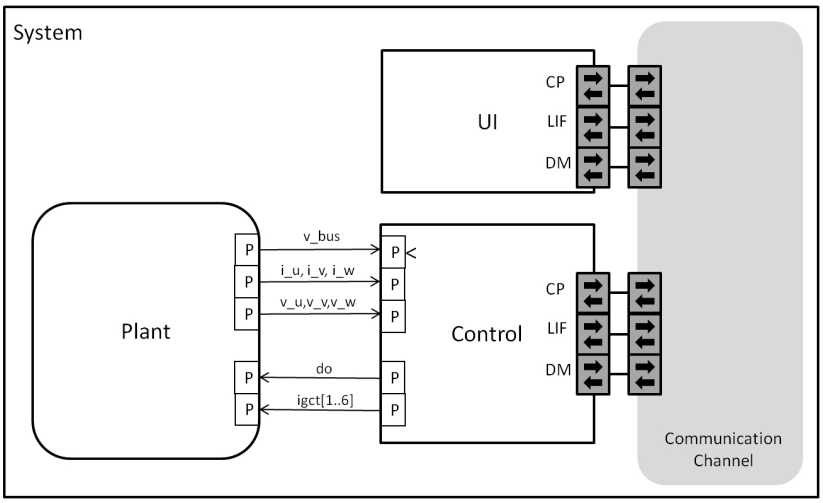
- Plant: The plant provides a simplified model of the plant, either as a CT or DT model. The plant interchanges signals with the control DAS using SystemC signals (sc_signal). The CT model is implemented using SystemC-AMS using electrical linear networks MoC.

Resource names

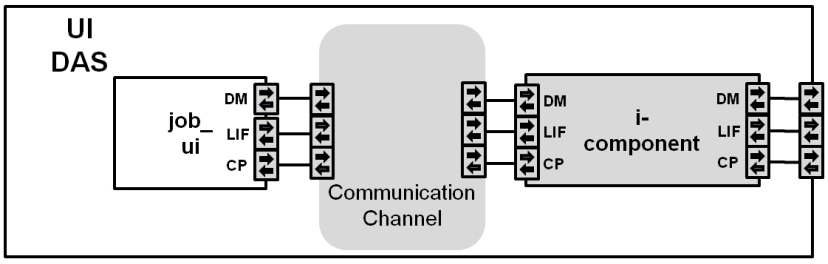
Table 7.1 lists the resource names of all components and real-time entities.

Resource Name	**Description**
rn:das:ui	User Interface (UI) DAS
rn:das:control	Control DAS
rn:job:ui.job_ui	User Interface (UI) job
rn:job:control.job_acq	Acquisition job
rn:job:control.job_control	Control job
rn:job:control.job_do	Digital output actuation job
rn:job:control.job_mod	Modulation job
rn:rte:ctrl_cmd	Control command identifier [enumeration]
rn:rte:ctrl_state	Control state identifier [enumeration]
rn:rte:v_bus	Bus voltage [V]
rn:rte:v_u	Voltage phase U [V]
rn:rte:v_v	Voltage phase V [V]
rn:rte:v_w	Voltage phase W [V]
rn:rte:i_u	Current phase U [A]
rn:rte:i_v	Current phase V [A]
rn:rte:i_w	Current phase W [A]
rn:rte:do	Digital output command [on/off]

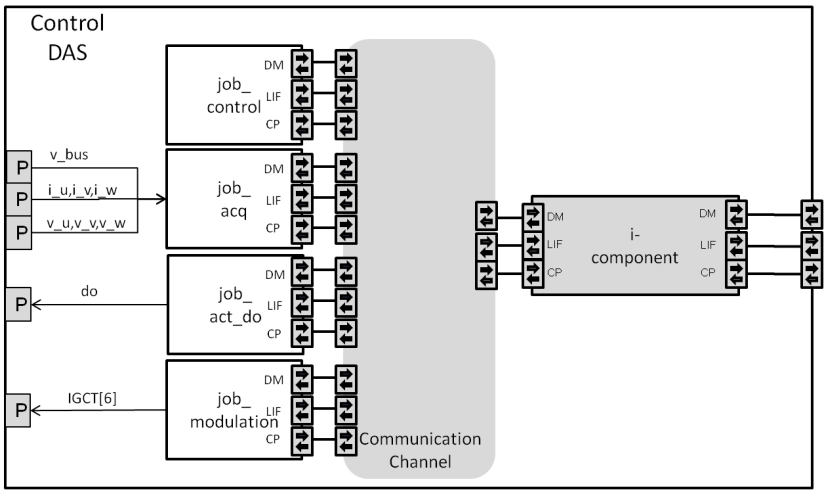
Table 7.1: System resource names, components and real-time entities.



(a) System



(b) UI DAS



(c) Control DAS

Figure 7.5: Example industrial real-time control system.

Timing configuration

Selected model timing configuration is described in Table 7.2 where one sparse-time macrotick corresponds to one global time tick $\Gamma = tick = 2.5\ ms$ ($K = 1$) and $\mu tick = 25\ \mu s$.

- The plant is executed with a fixed step-size of one $\mu tick$ (25 μs).
- The acquisition and modulation jobs are executed with a period of one global time tick (2.5 ms).
- The user interface, control and digital output actuation jobs (job_ui, $job\_control$, job_act_do) are executed with a period of $p = 200\ ms$ and different offsets. At offset 10 %, oversampled acquisition values are procesed by the control job generating set-points for the modulator job and the digital output actuation job to be executed with an offset of 20 %. At offset 50 %, the user interface job is executed providing updated information and receiving commands from the end user.

Figure 7.6 provides a time-cycle graphical representation where UI job simulation instant is represented with dashed lines, control jobs simulation instants are represented with black lines and plant simulation instant is represented with gray lines. As the plant is executed 8.000 iterations per period ($k = 8000 = 200\ ms/25\ \mu s$) the circle seems to be filled gray.

DAS	**job**	**period (p)**	**phase (ϕ)**
UI	0, ui	$200\ ms = 80 \cdot \Gamma$	50 %
Control	1, acq	$2.5\ ms = 1 \cdot \Gamma$	0 %
	2, control	$200\ ms = 80 \cdot \Gamma$	10 %
	3, do	$200\ ms = 80 \cdot \Gamma$	20 %
	4, mod	$2.5\ ms = 1 \cdot \Gamma$	0 %
Plant	5, plant	$25\ \mu s = 1 \cdot \mu tick$	0 %

Table 7.2: System timing configuration.

Metrics

The example has been implemented as a C++ application based on mmE-TTM and SystemC. The main metrics and figures are listed as follows:

- Centralized model (CT plant) source code: 16 source code files, 8 header files (.h) and 8 source files (.cpp,.c), 12 classes and 2.700 lines of code from which 681 are statements.

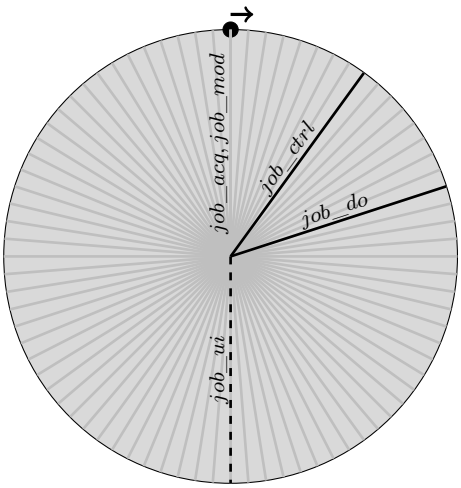


Figure 7.6: System timing configuration represented as time cycle.

- Centralized model (DT plant) source code: 36 source code files, 18 header files (.h) and 18 source files (.cpp,.c), 18 classes and 4.591 lines of code from which 1.040 are statements.
- Centralized model (DT plant) SystemC-AMS statistics: 1 dataflow cluster, 6 dataflow modules/solver and 2.500 ns cluster period. Linear solver instance with 25 equations for 24 modules and 2.500 ns initial time step.

7.2.3 Experimental evaluation

Table 7.3 describes selected experiments to be simulated and evaluated. In a distributed model, one submodel integrates the UI DAS and the second submodel integrates the plant and the control DAS.

Expected Outcome and Hypotheses

Simulation time trace file(s) should be identical for all experiments, all components must be executed at the same simulation time macroticks as defined by the timing configuration. Simulation time and physical time can be coupled by a given ratio if the simulation platform provides enough computational performance. The coupling will not be perfect and the execution time jitter will depend on the simulation platform.

Network protocol analyser data will be available for distributed topologies based on Ethernet or TTE communication protocols, except for the multicore distributed model because Ethernet communication is local. The communication time jitter for TTE should be in the range of microseconds [SGAK06].

ID	Simulation Platform	Model	Protocol	Plant MoC	Time
1	Single	Centralized	None	*DT*	coupled x1.0
2	Single	Centralized	None	*DT*	coupled x0.5
3	Single	Centralized	None	*DT*	coupled x2.0
4	Single	Centralized	None	*DT*	decoupled
5	Single	Centralized	None	*CT*	coupled
6	Single	Centralized	None	*CT*	decoupled
7	Single multicore	Distributed	Ethernet	*DT*	coupled x1.0
8	Single multicore	Distributed	Ethernet	*DT*	coupled x0.5
9	Single multicore	Distributed	Ethernet	*DT*	coupled x2.0
10	Single multicore	Distributed	Ethernet	*DT*	decoupled
11	Distributed	Distributed	Ethernet	*DT*	coupled x1.0
12	Distributed	Distributed	Ethernet	*DT*	decoupled
13	Distributed	Distributed	TTE	*DT*	coupled x1.0
14	Distributed	Distributed	TTE	*DT*	coupled x0.5
15	Distributed	Distributed	TTE	*DT*	coupled x2.0

Table 7.3: Analyzed simulation experiments.

Centralized Model

The centralized model shown in Figure 7.5(a) is executed in a single computer (Figure 7.1(a)), using the following experimental setup options: CT / DT plant MoC and coupled / decoupled time. The simulation times trace file(s) are identical for all experiments. Table 7.4 and Figure 7.7 describe time results of interest.

- Experiment 1 simulates the *DT* model coupled with physical time with a ratio $r_{sync} = 1.0$ as shown in Figure 7.7(a). The simulation execution jitter, shown in Figure 7.7(b), has an average value of 4 μsec, maximum value of 0.3 ms and variance value of $1.78e-11$. The jitter peak seems to be due to an sporadic Windows operating system task switch.

- Experiment 2 simulates the *DT* model coupled with physical time with a ratio $r_{sync} = 0.5$ as shown in Figure 7.7(a). The simulation execution jitter has an average value of 8 μsec, a maximum value of 9.32 ms and variance value of $4.83e-07$.

- Experiment 3 simulates the *DT* model coupled with physical time with a ratio $r_{sync} = 2.0$ as shown in Figure 7.7(a). The simulation execution jitter has an average value of 4 μsec, maximum value of 0.3 ms and variance value of $1.15e-11$.

- Experiment 4 simulates the *DT* model decoupled from physical time as shown in Figure 7.7(c) (accelerated simulation).

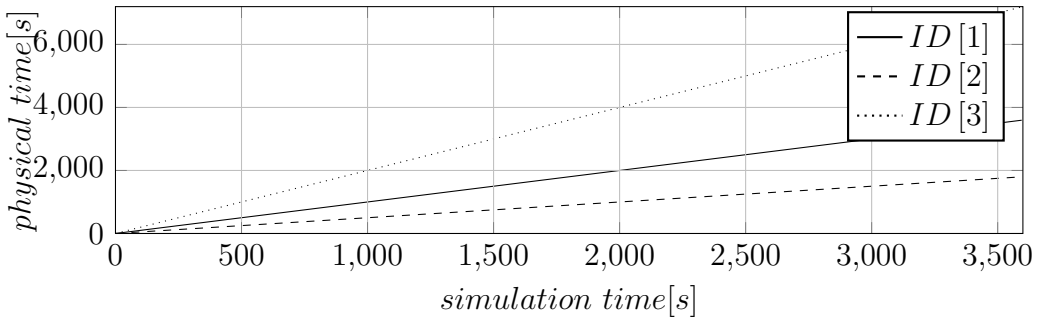
- Experiment 5, the laptop does not have enough computational performance to execute the *CT* model coupled with physical time, thus the result is considered *Not Applicable* (N/A).

- Experiment 6 simulates the *CT* model decoupled from physical time.

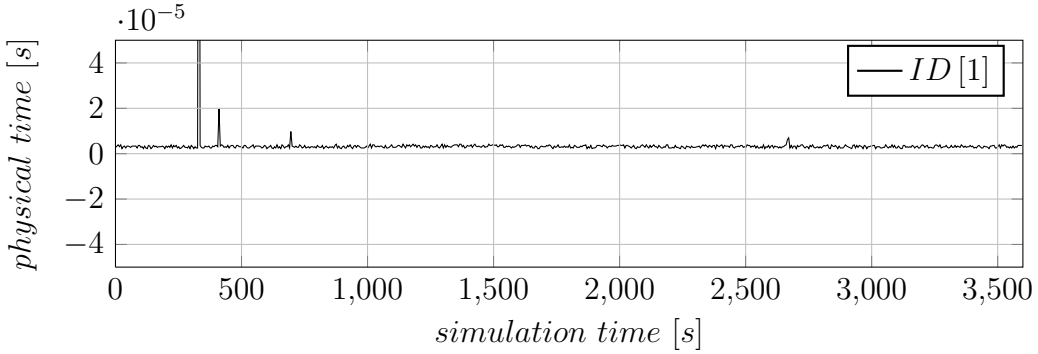
ID	**Platform(s)**	**Simulation Time**	**Execution Time**	**Ratio**
1	Single	3600.00 *sec*	3600.00 *sec*	$r_{sync} = 1.000$
2			1800.00 *sec*	$r_{sync} = 0.500$
3			7200.00 *sec*	$r_{sync} = 2.000$
4			134.15 *sec*	$r_{avg} = 0.038$
5			N/A	N/A
6			59424.68 *sec*	$r_{avg} = 16.506$

Table 7.4: Centralized model simulation time results.

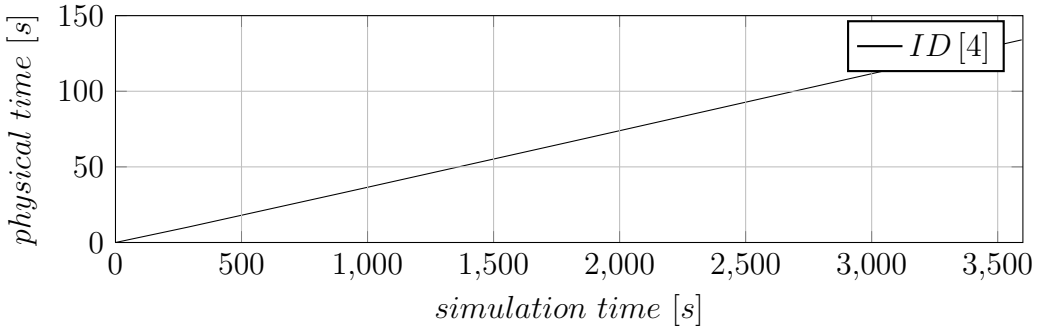
Figure 7.8 shows the simulation results of interest, while enable signal is active the control commands the plant and generates a triphasic sinusoidal voltage of configurable amplitude and frequency and when the enable signal is not active, the output voltage is zero.



(a) Execution time for experiments $1, 2$ *and* 3.

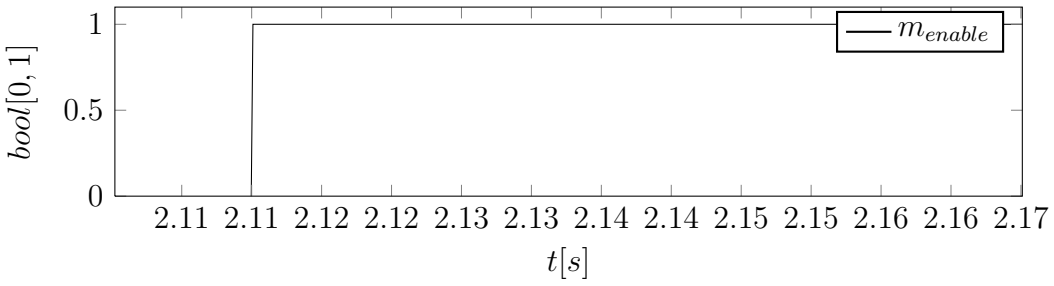


(b) Execution time jitter for experiment 1.

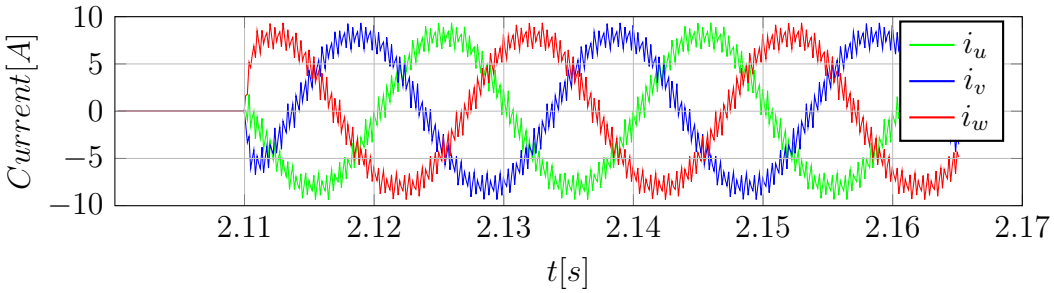


(c) Execution time for experiment 4.

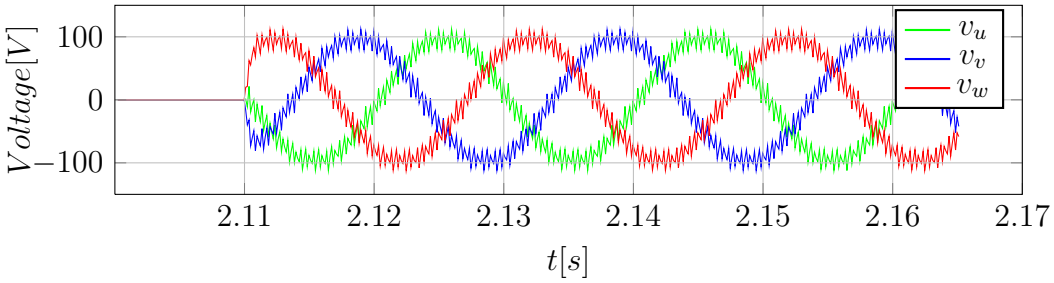
Figure 7.7: Centralized model execution time.



(a) Modulator enable command.



(b) Phase currents: i_u, i_v, i_w.



(c) Phase voltages: v_u, v_v, v_w.

Figure 7.8: Centralized model simulation results with CT model of the plant.

Distributed Model - Ethernet Multicore

The distributed model shown in Figure 7.9 is executed in the distributed computer (Figure 7.1(b)), using the following experimental setup options: DT plant MoC, Ethernet protocol and coupled / decoupled time. The model is partitioned in two submodels, as shown in Figure 7.9, one submodel integrates the UI DAS and the second submodel integrates the plant and the control DAS. Each model partition is executed in a different executable file instance and assigned by Windows XP to a different core where both models communicate among them using Ethernet. Simulation times trace file(s) are identical for all experiments and Table 7.5 and Figure 7.10 describe time results of interest.

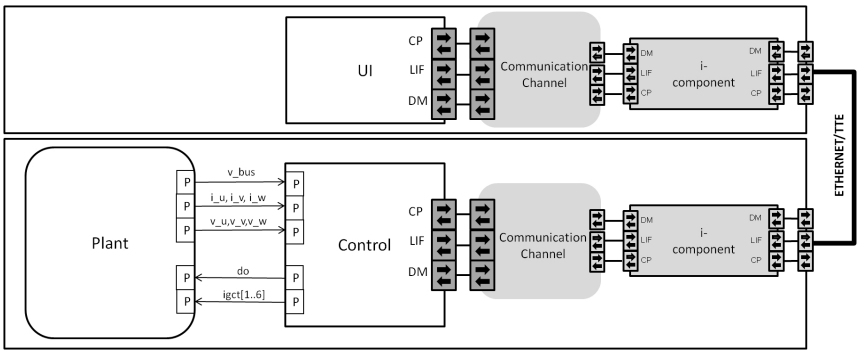


Figure 7.9: Distributed model, example industrial real-time control system.

ID	Platform(s)	Simulation Time	Execution Time	Ratio
7	Single multicore	3600.00 *sec*	3600.00 *sec*	$r_{sync} = 1.000$
8			1800.00 *sec*	$r_{sync} = 0.500$
9			7200.00 *sec*	$r_{sync} = 2.000$
10			118.93 *sec*	$r_{avg} = 0.033$

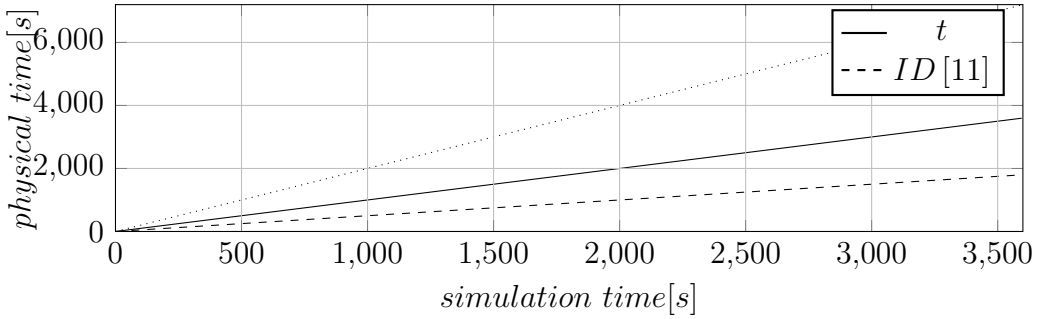
Table 7.5: Distributed model simulation time results (multicore and Ethernet).

- Experiment 7 simulates the *DT* model coupled with physical time with $r_{sync} = 1.0$ as shown in Figure 7.10(a). The simulation execution jitter, shown in Figure 7.10(b), has an average value of 4 μsec, maximum value of 0.35 ms and variance value of $2.09e-11$. Execution jitter peak seems to be due to sporadic Windows operating system task switch.

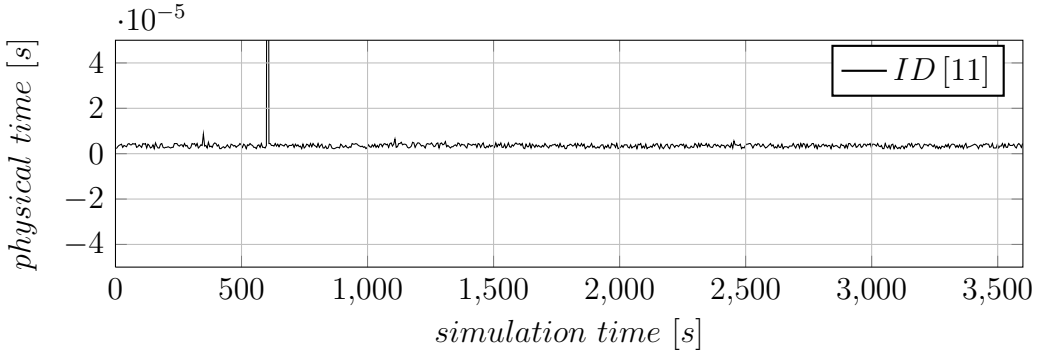
- Experiment 8 simulates the *DT* model coupled with physical time with $r_{sync} = 0.5$ as shown in Figure 7.10(a). The simulation execution jitter has an average value of 4 μsec, maximum value of 0.32 ms and variance value of $2.13e-11$.

- Experiment 9 simulates the *DT* model coupled with physical time with $r_{sync} = 2.0$ as shown in Figure 7.10(a). The simulation execution jitter has an average value of 4 μsec, maximum value of 0.37 ms and variance value of $1.54e-11$.

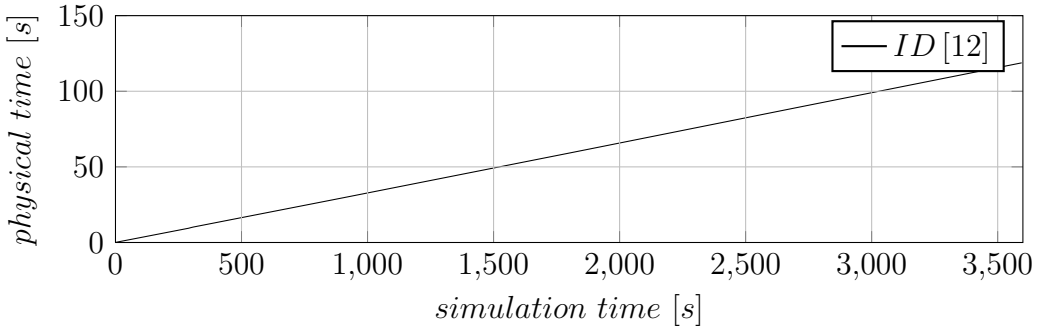
- Experiment 10 simulates the *DT* model decoupled from physical time as shown in Figure 7.10(c) (accelerated simulation). For the analysed model, the distributed execution time is less than the centralized execution time (compare experiments 4 and 10).



(a) Execution time for experiment 5.



(b) Execution time jitter for experiment 5.



(c) Execution time for experiment 6.

Figure 7.10: Distributed model time results (multicore Ethernet).

Distributed Model - Ethernet

The distributed model shown in Figure 7.9 is executed in two computers communicated using Ethernet as shown in Figure 7.1(b). Simulation times trace file(s) are identical for all experiments and Table 7.6 and Figure 7.11 describe time results of interest.

- Experiment 11 simulates the *DT* model coupled with physical time with

ID	Platform(s)	Simulation Time	Execution Time	Ratio
11	Distributed	3600.00 *sec*	3600.00 *sec*	$r_{sync} =$ 1.000
12			49.20 *sec*	$r_{avg} =$ 0.013

Table 7.6: Distributed model simulation time results (Ethernet).

$r_{sync} = 1.0$ as shown in Figure 7.11(a). The communication execution jitter, shown in Figure 7.11(b), has an average value of 15 μsec, maximum value of 40μ *sec*) and variance value of $2,05997e-10$.

- Experiment 12 simulates the *DT* model decoupled from physical time as shown in Figure 7.11(c) (accelerated simulation). For the analysed model, the distributed execution time is much less than the centralized execution time (compare experiments 4 and 12).

Distributed Model - TTE

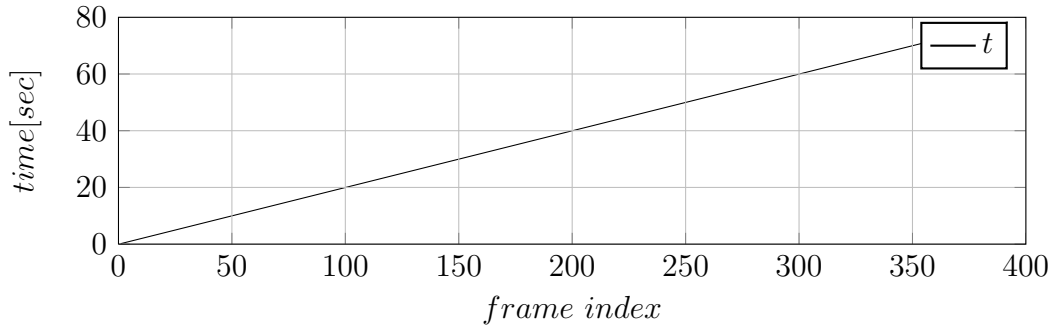
The distributed model shown in Figure 7.9 is executed in two computers communicated using TTE as shown in Figure 7.1(b). Simulation times trace file(s) are identical for all experiments and Table 7.7 and Figure 7.12(b) describe time results of interest.

ID	Platform(s)	Simulation Time	Execution Time	Ratio
13	Distributed	3600 *sec*	3600 *sec*	$r_{sync} = 1.000$
14			1800 *sec*	$r_{sync} = 0.500$
15			7200 *sec*	$r_{sync} = 2.000$

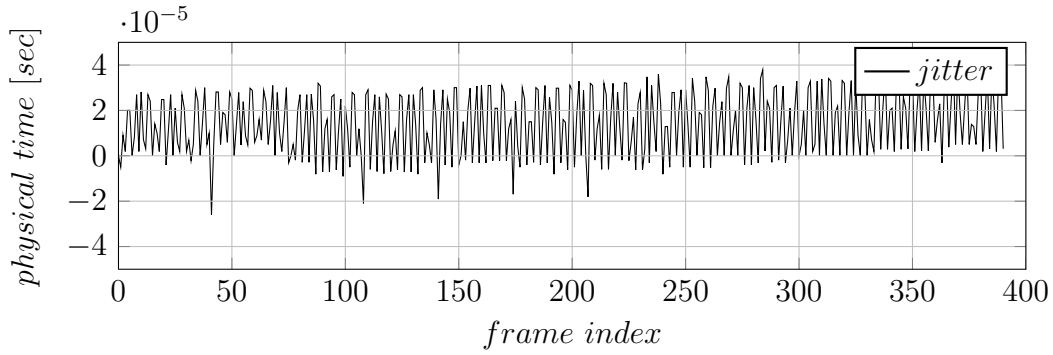
Table 7.7: Distributed model simulation time results (TTE).

- Experiment 13 simulates the *DT* model coupled with physical time with $r_{sync} = 1.0$ as shown in Figure 7.12(a). The simulation communication jitter, shown in Figure 7.12(b), has an average value of 8 μsec, maximum value of 14 μsec and a variance value of $5.38e-12$.

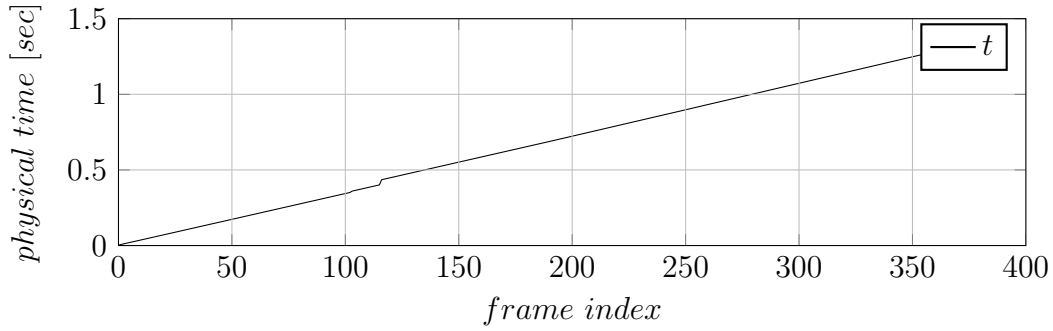
- Experiment 14 simulates the *DT* model coupled with physical time with $r_{sync} = 0.5$ as shown in Figure 7.12(a). The simulation communication jitter has an average value of 13 μsec, maximum value of 54 μsec and a variance value of $5.12e-9$.



(a) Communication time for experiment 11.



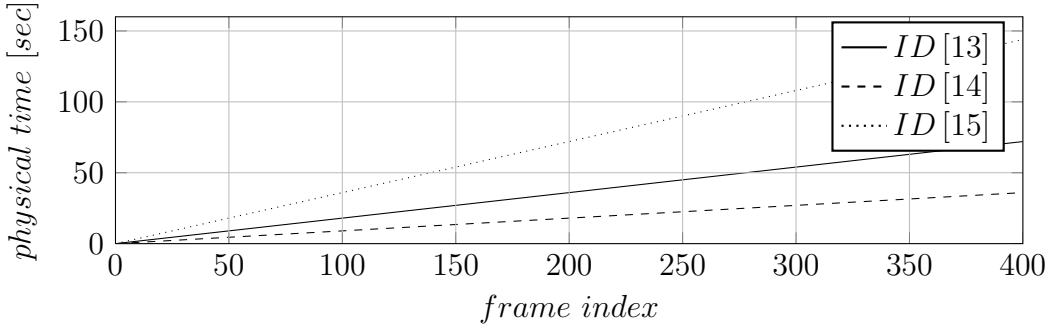
(b) Communication time jitter for experiment 11.



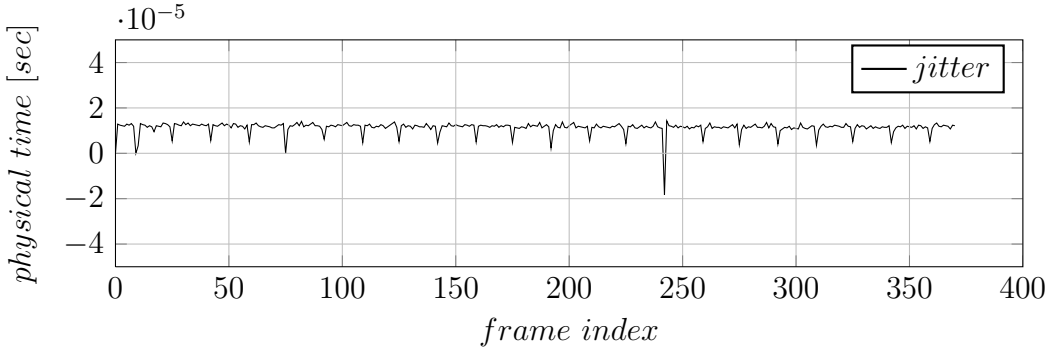
(c) Communication time for experiment 12.

Figure 7.11: Distributed model communication time for Ethernet.

- Experiment 14 simulates the *DT* model coupled with physical time with $r_{sync} = 2.0$ as shown in Figure 7.12(a). The simulation communication jitter has an average value of 10 μsec, maximum value of 25 μsec and a variance value of $2.81e - 11$.



(a) Communication time for experiments 13, 14 *and* 15.



(b) Communication time jitter for experiment 13.

Figure 7.12: Distributed model communication time for TTE.

7.2.4 Conclusion

The real-time control system has been designed using the mmE-TTM and xfE-TTM described in Section 5 and experimentally evaluated with different simulation topologies leading to the following conclusions:

- Simulation time determinism property of the model has been confirmed as expected, simulation trace file(s) are identical for all experiments.
- Simulation data determinism has not been confirmed because required data determinism model assumption described in Section 5.5 has not been met for floating-point operations. Selected compiler options (see Section 7.1.1) and compiler default mathematical libraries do not provide floating-point bit-exact results among platforms.
- Open simulations coupled with physical time have an execution and communication jitter dependent of the simulation platform and communication network. Whenever possible the jitter should be small and bounded (e.g., TTE provides a small and bounded jitter [SGAK06]).

- The selection between CT and DT based plant models is a simulation execution time vs. accuracy trade off. In this case-study the DT based plant model is executed 434 times faster (16.506/0.038 = 434.36) than the CT based plant model at the cost of simulation accuracy. Simulation execution time results might be considered reasonable for the design of safety-critical embedded systems, where DT based plant models can be used for faster intermediate simulation assessments while slower CT based plant models can be used for final simulation assessments.

- Simulation platform operating-system, compiler tools and compiler options have a considerable effect on the simulation execution time. In this case-study Ubuntu based simulation platforms used for distributed platform experiments have considerable lower simulation execution time than Windows based simulation platforms (Experiment 12 vs. Experiment 10, 118.93 vs. 49.20 sec). On the other hand, as it could be generally expected given the same simulation platform the distributed simulation execution time is smaller than the centralized one (Experiment 4 vs. Experiment 10, 134.15 vs. 118.93 sec). Nonetheless, this assumption requires a proper model partition and cannot be generalised.

7.3 ETCS Odometry

The second case study models an example odometry safety-critical embedded system (SIL-4), which is subject to Simulated Fault Injection (SFI) using a generic SystemC based SFI environment described in [PAAP10]. The odometry system provides traveled speed and distance estimations using diverse redundant sensors (e.g., radars, accelerometer and wheel sensors) and sensor-fusion algorithm [MAR+08].

7.3.1 System description

The *European Rail Traffic Management System* (ERTMS) [Win09] is an European Union backed initiative for the definition of a unique train signaling standard throughout Europe, where the on-board *European Train Control System* (ETCS) is an *Automatic Train Protection* (ATP). The high-speed train on-board ETCS is a safety-critical embedded system (SIL-4) that protects the train by supervising the traveled distance and speed, activating the emergency brake if authorized values are exceeded. The ETCS safety-computer is called *European Vital Computer* (EVC).

The ETCS relies on the distance and speed measurements provided by the on-board odometry system, which performs dead reckoning based on a set of diverse sensors such as wheel angular speed encoders, longitudinal accelerometers and Doppler radars. The standard requires the odometry precision to be within the boundaries described in Equation 7.1, the error must always be lower than five meters plus five percent of the traveled distance for a maximum traveling speed of $500km/h$, where s is the traveled distance and s_m is the measured distance. The railway infrastructure provides train absolute positions whenever a new eurobalisse is read, and this location is used to correct an recalibrate on-line the odometry system.

$$\forall t, |s_m(t) - s(t)| \leq 5m + (5/100) \cdot s(t) \tag{7.1}$$

Reference architecture

Figure 7.13 shows the ETCS on-board reference architecture [Win09] partitioned into several subsystems all connected to the EVC safety computer:

- The EVC is the central safety processing unit that communicates with all subsystems and executes all safety functions associated to the traveling speed and distance supervision. The EVC executes the safety kernel and includes the odometry subsystem, which estimates the traveling distance and speed based on a set of diverse sensors.
- The *Driver Machine Interface* (DMI) is the driver interface, periodically updated with state parameters (e.g., traveling speed) and transmitting sporadic event information (e.g., button pressed).
- The *Juridical Recorder Unit* (JRU) must record all relevant external events (e.g., new eurobalisse message) and internal events (e.g., activate emergency brake).
- The *Balise Transmission Module* (BTM) telepowers eurobalisses as the train passes them, receives information sent by the eurobalisse and transmits the demodulated information to the EVC. The *Loop Transmission Module* (LTM) provides equivalent functionality with received information from Euroloops.
- The *Global System for Mobile Communications - Railway* (GSM/R) interface enables the bidirectional information exchange between remote control centers and the train, supporting track and operation related data exchange.

- The *Train Interface Unit* (TIU) reads / writes a set of input / output digital values, some of which are safety related such as the emergency brake digital output. In addition to this, additional train interfaces such as *Multifunction Vehicle Bus* (MVB) must be considered.

- Odometry sensors such as encoders, Doppler radars and longitudinal accelometers provide measurements of diverse nature such as pulses with a frequency proportional to the longitudinal speed, speed measurement based on Doppler effect and longitudinal acceleration.

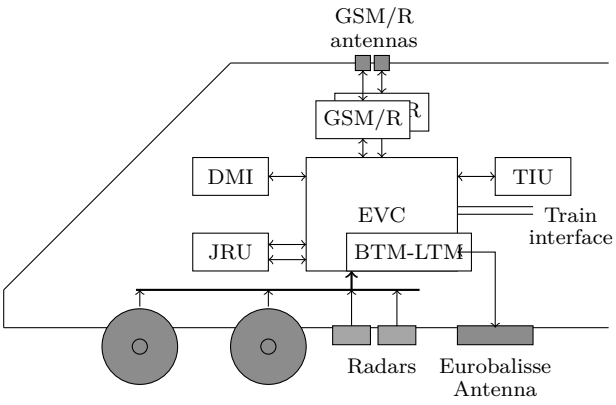


Figure 7.13: ETCS on-board reference architecture [Win09].

System architecture design

Figure 7.14 shows the simplified architecture design of a high-speed train on-board ETCS solution proposal. TTE is used as the single communication infrastructure for the interconnection of multiple subsystems of mixed criticality and timing constraints: *Triple Modular Redundancy* (TMR) of computers that form the EVC, DMI, GSM/R, JRU and BTM.

The odometry sensor-fusion algorithm to be used requires at least the availability of two encoders and one accelerometer. A periodic input agreement protocol is executed at the EVC, between host computers, with the periodic publication of all sensor measurements so that all host computers compute odometry with the same input values. Two encoders and one longitudinal accelerometer sensors are connected to each EVC host computer, so that the failure of any single host computer or multiple sensors does not jeopardize availability while ensuring safety.

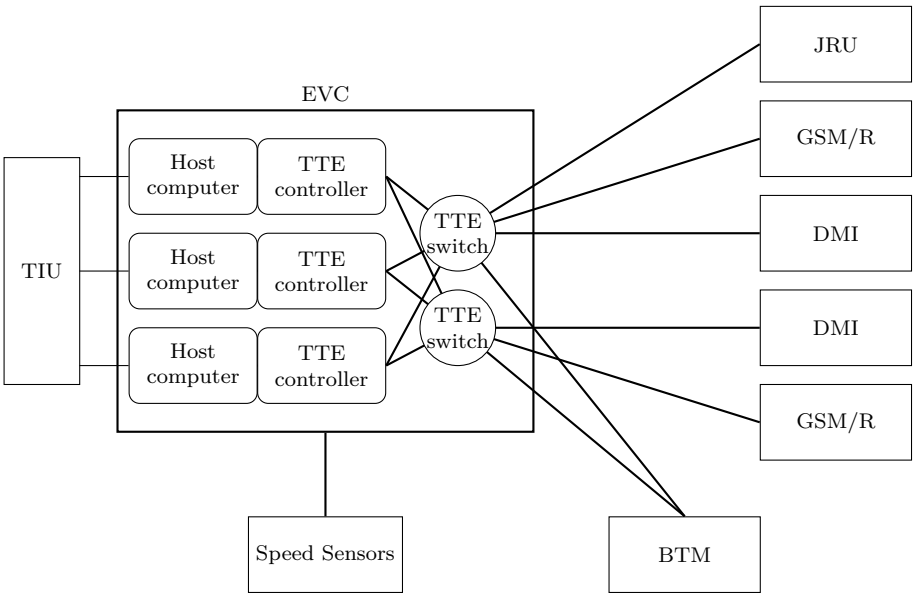


Figure 7.14: ETCS design.

7.3.2 System design

As shown in Figure 7.16, the odometry subsystem is composed of three replicated odometry DASes, one instance per EVC host computer. The odometry DAS is composed of four jobs as shown in Figure 7.17:

- The acquisition job (job_acq) reads input sensors and performs input agreement protocol among replicas.
- The calculation job (job_calc) uses the previously acquired sensor information to calculate using a sensor-fusion algorithm [MAR+08] the estimated traveled distance (sm), speed (vm) and acceleration (am).
- The resynchronization job (job_resync) receives resynchronization information from the EVC, the absolute position reading (s, ts) used to resynchronize the speed and distance calculation algorithm parameters. The absolute position reading is received aperiodically, whenever a balise is passed.
- The voter job (job_voter) performs output agreement protocol of calculated odometry estimations with other replicas. The voter output is the agreed odometry estimation of the traveled distance (smv), speed (vmv) and acceleration (amv).

The system design is integrated with a generic SystemC based SFI environment [PAAP10], which enables the early dependability assessment of the design by means of SFI. The selection of faults to be considered is guided by the FMEA analysis where potential faults that could lead to system failure are analyzed at different abstraction levels. Figure 7.15 shows the design model and fault injection environment, identifying the following main modules:

- The system interacts with the environment, composed by the physical world on which the odometry operates (train and railway) and the infrastructure the odometry interacts with (ETCS subsystems).
- The odometry harness, provides saboteur encoder and longitudinal sensors connected to the odometry system.
- The checker module verifies a set of rules during system simulation.
- The fault injection module injects faults in the model.

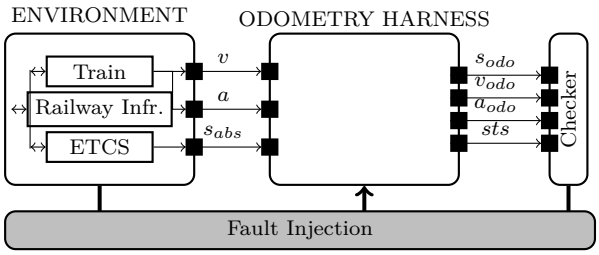


Figure 7.15: Design model and fault injection environment.

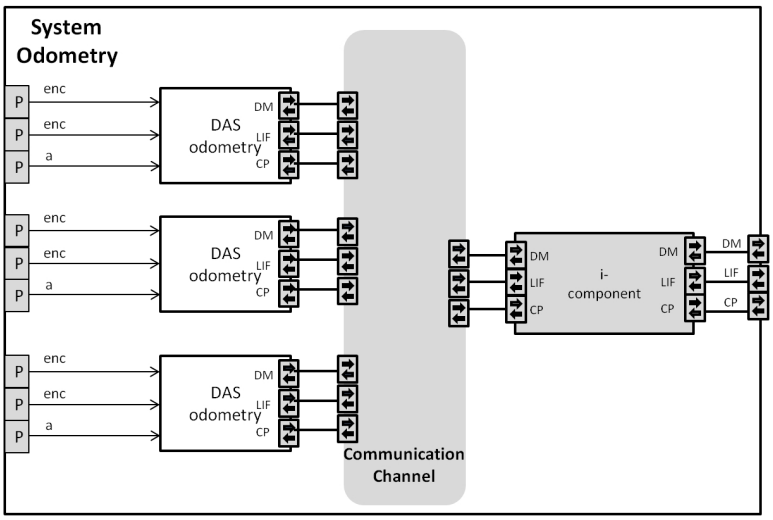


Figure 7.16: Example odometry system model.

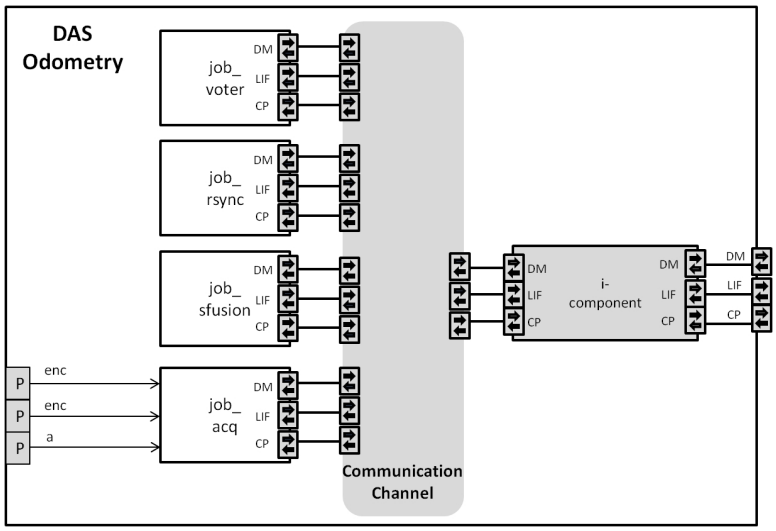


Figure 7.17: Example odometry DAS model.

Resource names

Table 7.8 lists the resource names of all components and real-time entities.

Resource Name	Description
rn:das:odometry[0]	Odometry DAS instance A
rn:das:odometry[1]	Odometry DAS instance B
rn:das:odometry[2]	Odometry DAS instance C
rn:job:odometry[x].job_acq	Acquisition job
rn:job:odometry[x].job_calc	Calculation job (sensor-fusion)
rn:job:odometry[x].job_resync	Resynchronization job
rn:job:odometry[x].job_voter	Voter job
rn:rte:odometry[x].s	External absolute distance measurement [m]
rn:rte:odometry[x].ts	External absolute distance measurement time instant [ms]
rn:rte:odometry[x].sm	Distance estimation [m]
rn:rte:odometry[x].vm	Speed estimation [m/s]
rn:rte:odometry[x].am	Acceleration estimation [m/s]
rn:rte:odometry[x].smv	Agreed (voted) distance estimation [m]
rn:rte:odometry[x].vmv	Agreed (voted) speed estimation [m/s]
rn:rte:odometry[x].amv	Agreed (voted) acceleration estimation [m/s]

Table 7.8: System resource names, components and real-time entities.

Timing configuration

Table 7.10 shows timing constraints and Table 7.9 shows the odometry subsystem timing configuration, based on [MTAC01]. In this example, one sparse-time macrotick corresponds to one global time tick $\Gamma = tick = 100\ ms$ ($K = 1$) and in order to execute the plant with higher time resolution a local fixed-time step $\mu tick = 2\ ms$ is specified.

Figure 7.18 provides a time-cycle graphical representation where job_rsyn, job_calc and job_vote simulation time instants are represented with black lines and job_acq and plant simulation time instants are represented with gray lines.

DAS	**job**	**period** (p)	**phase** (ϕ)
Train Plant	0, plant	$2\ ms = 1 \cdot \mu tick$	0 %
Odometry	1, acq	$2\ ms = 1 \cdot \mu tick$	0 %
	2, rsyn	$100\ ms = 1 \cdot \Gamma$	0 %
	3, calc	$100\ ms = 1 \cdot \Gamma$	50 %
	4, vote	$100\ ms = 1 \cdot \Gamma$	75 %

Table 7.9: ETCS odometry model timing configuration.

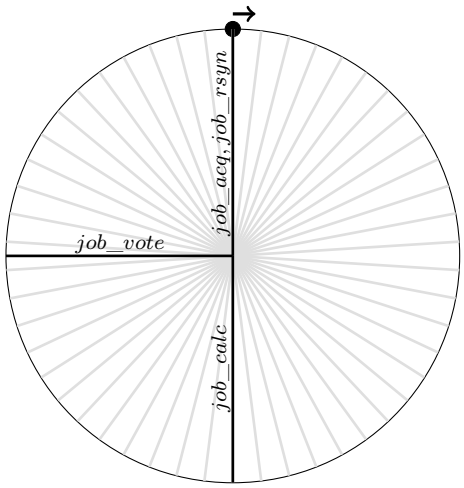


Figure 7.18: System timing configuration represented as time cycle.

7.3.3 Experimental evaluation

This section describes the simulation results of previously designed odometry safety system within the SFI environment shown in Figure 7.15, as already analyzed in [PAAP10]. Table 7.11 describes the faults to be inserted, where

Constraint	Description
$S\,(DAS\_odo[0..2])$	Odometry replicated DAS channels must be executed synchronously in order to get a synchronous reading of all input sensors.
$P\,(job\_acq, job\_calc)$ $P\,(job\_resync, job\_calc)$	Calculation job must be executed after acquisition job and resynchronization job in order to ensure calculation data consistency.
$H\,(job\_calc \vert job\_acq)$	Acquisition job performs oversampling, thus, it is executed with higher frequency than the calculation job. The harmonicity is 50, 100 $ms/2\ ms$.
$F\,(s \vert as) \leq 100\ ms$	The traveled distance calculation is aperiodically resynchronized with the absolute position reading from a balise. The freshness of the absolute position state information limits the odometry resynchronization accuracy, and this way it is bounded to a reasonable value of one execution period.

Table 7.10: ETCS odometry model timing constraints.

the notion of time tick is replaced by the notion of distance (s) and faults are inserted at a given traveled distance tick with a granularity of $1km$.

Experiment

The odometry system is subject to a simulated fault injection experiment based on the simulator commands listed in Table 7.11. A common assessment scenario and fault environment is described, which defines the operating train speed setpoints (v_{sp}) and default values for accelerometer location faults (ζ, θ), distance between eurobalisses ($s_{balisse}$), wheel adhesion factor ($adhesion$) and wheel radius difference ($\Delta wheel\_radius_{diff}$). The railway adherence factor ($adhesion$) behavioral fault is updated at different distance windows.

Simulator commands listed in Table 7.11 are used by the transactor and system model saboteurs as identified in Table 7.12. Fault tolerance is achieved using replication and error masking, by the external voter and the sensor-fusion algorithm that relies on a set of diverse redundant sensors. The checker module generates a trace file with variables of interest and verifies a set of rules during system simulation (e.g., Equation 7.1).

Expected Outcome and Hypotheses

Determinism is a sufficient precondition for logical reasoning [Kop08a]. Simulation time determinism property of the design model under analysis should

window(distance)	idx	value
$-1, -1, -1$	$s_{balisse}$	2.5 [km]
$-1, -1, -1$	$adhesion$	100 [%]
$-1, -1, -1$	θ	2 [grad]
$-1, -1, -1$	ζ	2 [grad]
$-1, -1, -1$	$\Delta wheel\_radius_{diff}$	0.001 [m]
$0, 0, 2$	v_{sp}	15 [km/h]
$0, 2, 3$	v_{sp}	50 [km/h]
$0, 5, 95$	v_{sp}	250 [km/h]
$0, 100, 50$	v_{sp}	125 [km/h]
$0, 150, 100$	v_{sp}	250 [km/h]
$0, 250, 250$	v_{sp}	15 [km/h]
$0, 275, F$	v_{sp}	0 [km/h]
$0, 2, 200$	$adhesion$	80 [%]
$0, 50, 150$	$adhesion$	70 [%]

Table 7.11: Experiment command table.

idx	Used by
v_{sp}	Transactor
$s_{balisse}$	Transactor
$\Delta wheel\_radius_{diff}$	Transactor
θ	Accelerometer sensor saboteur
ζ	Accelerometer sensor saboteur
$adhesion$	Encoder sensor saboteur

Table 7.12: Simulator commands used by transactor and saboteurs.

support the early dependability assessment by means of SFI, where faults can be injected at precise time instants and implemented fault tolerance mechanisms validated.

Experiment results

The high-speed train odometry system is subject to SFI with the previously described experiment leading to the results illustrated in Figure 7.19. Simulations are executed using a laptop with a dual core microprocessor (2.40 GHz) and Windows XP operating system. The simulated two hours (7200 sec) and 290 km long journey is executed in less than 60 seconds.

Small speed measurement errors can be identified in Figure 7.19 during acceleration and braking phases due to wheel slid-skid. Speed measurement errors lead

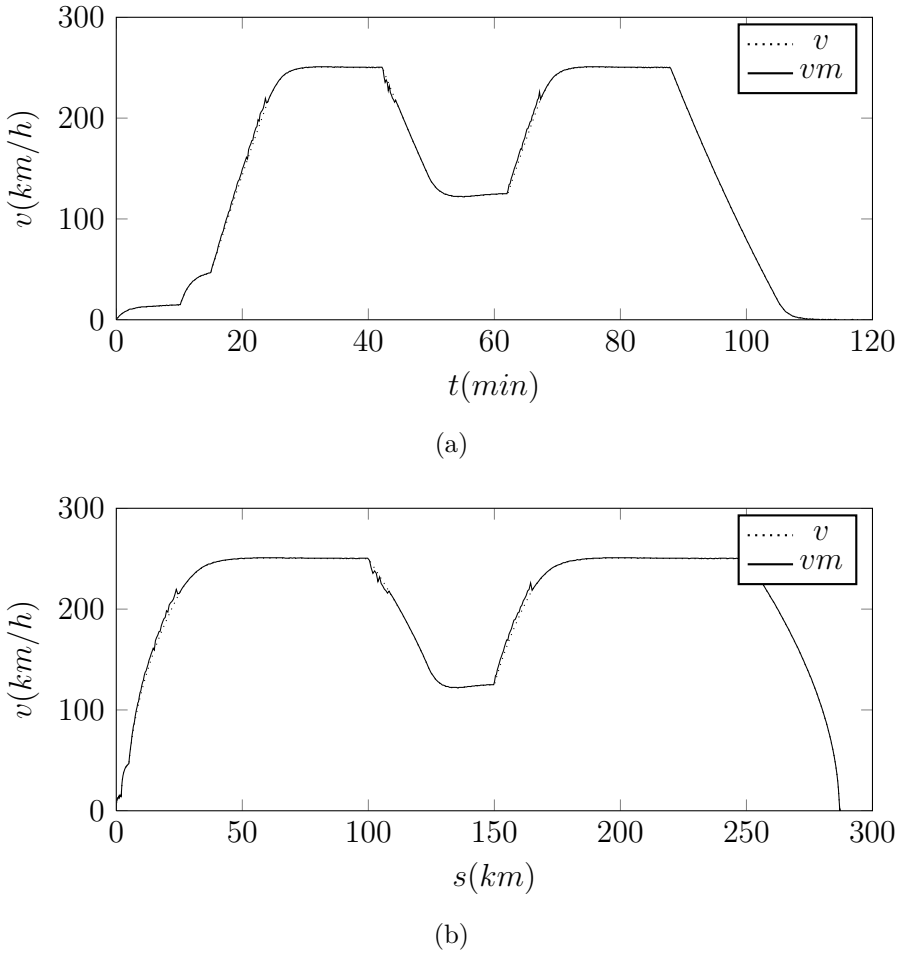


Figure 7.19: Train speed vs. measured speed.

to distance measurement errors, where voted measurement errors are within the required safety margin described in Equation 7.1 as verified by the checker module. Each odometry DAS employs sensor fault tolerance and sensor-fusion to provide an accurate and reliable measurement even under the presence of external environmental faults. On the other hand, highest distance errors are due to slid-skid faults originated by external environmental faults such as degradated adhesion factor and incorrect acceleration sensor positioning.

7.3.4 Conclusion

The ETCS odometry safety-critical embedded system has been designed using the mmE-TTM and xfE-TTM described in Section 5 and experimentally evaluated with a SFI based dependability assessment. Determinism is a sufficient

precondition for logical reasoning [Kop08a] and simulation time determinism property of the design model under analysis enables the early dependability assessment by means of SFI, where faults can be injected at precise time instants and implemented fault tolerance mechanisms validated.

All in all, the early dependability assessment has been successful and experimental results indicate that selected architectural fault tolerance mechanisms (TMR) and sensor-fusion provide sufficient fault tolerance for the given experiment. On the other hand, sensor fault tolerance and sensor-fusion algorithms could be further improved. For example, the addition of an additional diverse sensor such as a Doppler radar would enhance fault tolerance and mitigation during wheels slip-skid faults.

7.4 Summary

In order assess the proposed E-TTM modeling approach in a practical way, two case studies have been designed using the mmE-TTM and xfE-TTM described in Section 5 and experimentally evaluated: an example industrial real-time control system [PPO10] and a safety-critical odometry system [PAAP10].

The industrial real-time control system example (see Section 7.2) has been experimentally evaluated with different simulation topologies leading to the following conclusions summary:

- Simulation time determinism property of the design model under analysis has been confirmed as expected.
- Simulation data determinism has not been confirmed because required data determinism model assumption described in Section 5.5 has not been met for floating-point operations.
- Open simulations coupled with physical time have an execution and communication jitter dependent of the simulation platform and communication network. Whenever possible the jitter should be small and bounded.
- The selection between CT and DT based plant models is a simulation execution time vs. accuracy trade off, where DT based plant models can be used for faster intermediate simulation assessments while slower CT based plant models can be used for final simulation assessments.
- Simulation platform operating-system, compiler tools and compiler options have a considerable effect on the simulation execution time.

The ETCS odometry example (see Section 7.3) has been experimentally evaluated with a Simulated Fault Injection (SFI) based dependability assessment. Determinism is a sufficient precondition for logical reasoning [Kop08a] and simulation time determinism property enables the early dependability assessment by means of SFI, where faults can be injected at precise time instants and implemented fault tolerance mechanisms validated. This is indeed important for the development of safety-critical embedded systems because the international safety standard IEC-61508 highly recommends fault injection techniques in all steps of the development process, in order to analyze the reaction of the system in a faulty environment and to validate the correct implementation of fault tolerance mechanisms. SFI enables an early dependability assessment that reduces the risk of late discovery of safety related design pitfalls.

"Nihil est enim simul et inventum et perfectum"
(Nothing is ever invented and perfected at the same time)

CICERO, BRUTUS 71

Chapter 8

Conclusion

This chapter reviews the Executable Time-Triggered Model (E-TTM) approach, provides a critical analysis of the results and identifies future work lines.

8.1 Review

This thesis has presented the Executable Time-Triggered Model (E-TTM), a novel approach for the executable PIM modeling of safety-critical embedded systems based on Time-Triggered Architecture (TTA) . It can also be used in different development phases (see Chapter 3), for the generation of executable specifications and during the verification and validation phases using the model as 'reference model'.

Chapter 4 has analyzed the notion of time from different perspectives related to the development of safety-critical embedded systems and cognitive complexity reduction, identifying suitable time representations, time constraints expressiveness and time abstraction models to be included in the specification of the E-TTM meta-model.

E-TTM models are instantiations of the E-TTM meta-model (mmE-TTM) described in Chapter 5, which provides a global-simulation time and a strict separation of concerns (partition) between computation and communication, where components communicate among them by means of the exchange of messages across communication channels. The meta-model relies on a consistent notion of time, the sparse-time concept from the time-triggered MoC, that enables single or distributed models coupled or decoupled from physical time to be executed while preserving simulation time determinism (simulation time invariance). This consistent notion of time in conjunction with the ability to

express time and time constraints, also enables the preservation of time properties through model refinements steps and throughout the development process. The meta-model implementation is a C++ library that extends SystemC with the time-triggered Model of Computation (MoC) and enables the codesign and execution of models in SystemC.

As analyzed in Chapter 6, the E-TTM approach provides multiple potential benefits compared with state-of-the-art modeling solutions, but at the expense of targeting a specific architecture (TTA) and meeting a set of restrictive constraints. However, the developer could also use the E-TTM approach to model generic real-time and safety-critical embedded systems by relaxing the imposed constraints, and ensuring the consistency and preservation of time properties. In addition to this, E-TTM could also be used in conjunction with other modeling approaches such as MARTE, PFSM, synchronous languages, etc.

Finally Chapter 7 describes the development of two case-studies, two E-TTM models instantiated from the E-TTM meta-model using the SystemC based E-TTM execution framework.

8.2 Critical analysis of the results

This thesis provides an interesting approach towards the systematic development of 'correct by construction' safety-critical embedded systems based on the TTA. However, nothing is ever invented and perfected at the same time. A critical analysis could identify the following list of issues, where most issues could be addressed as future work as described in Section 8.3.

- The proposed E-TTM modeling approach targets the development of safety-critical embedded systems, but only the PIM level has been developed in this thesis.
  - The development of the PSM was outside the scope of the thesis but nonetheless already considered as possible future work (see 'PSM' in Section 8.3).
- The proposed E-TTM relies on the TTA, but not all safety-critical embedded systems are based on the TTA.
  - The E-TTM approach does not restrict the development process nor the underlying product architecture. If the underlying product architecture is TTA, properties of interest (e.g., time) could be systematically preserved throughout the development process, and if another architecture is used the designers are responsible for ensuring that properties of interest are preserved.

- This thesis focuses primarily on safety-critical embedded systems, and the constraints applied for the development of E-TTM might be too restrictive for the development of other types of embedded systems.
  - The E-TTM approach targets the development of safety-critical embedded systems and provides a solid time model foundation based on the sparse-time for this purpose. Nonetheless, it can also be used to model generic real-time and safety-critical embedded systems by relaxing the imposed constraints, in which case the developer is then responsible for ensuring the consistency and preservation of time properties.
- The development of safety-critical embedded systems requires the use of tools and formal methods to ensure the correctness of the product being developed. Neither the tools available nor the formal verification of E-TTM models have been described.
  - As described in Section 8.3 (see 'MBD'), there is an ongoing research effort towards the automatic SystemC code generation from multiple MBD languages such as UML and MARTE. So, a new MARTE profile could be defined to support the design of TTA based systems and automatic generation of E-TTM models.
  - The E-TTM enables code generated from formal languages (e.g., synchronous languages) to be integrated but so far it has not provided a formalism.
- The development and configuration (time and communication) of both case-studies has been manual. There is no tool support and therefore the generation of models and suitable configuration is quite tedious and error prone.
  - As previously described (see also 'MBD' in Section 8.3) a new MARTE profile could be defined to support the design of TTA based systems and automatic generation of E-TTM models.

8.3 Future work

The results and foundation of this thesis can be extended in various ways, leading to the identification of future work and research lines:

- Platform Specific Model (PSM): As explained in Section 3.2, at PSM level the designer maps the previously designed E-TTM PIM model into

physical distributed platforms, taking into account platform specific requirements and constraints. The definition of the E-TTM PSM level should also rely on the TTA architecture and address different integration levels, e.g., Time-Triggered Network-on-Chip (TTNoC) [OESHK08].

- Model-Based Design (MBD): A new MARTE profile could be defined to support the design of TTA based systems and automatic generation of E-TTM models. There is already an ongoing research effort towards the automatic SystemC code generation from multiple MBD languages such as UML and MARTE that could be used as reference.

- Tools: The availability of a MARTE profile for the modeling and generation of E-TTM models and a PSM view support, would enable the E-TTM modeling approach to be integrated using existing tools [wwwi] for the development of TTA based solutions. This would close the loop and enable the the design to be automated. PIM and PSM views, and bridge the gap to the final system implementation.

- TT-WSDL: Web Service Description Language (WSDL) is a standard language for the description of Web Services (WS) that describes how potential clients are intended to interact with the services [www07, W3C07]. The Q-WSDL [D'A06] extension proposed by D'Ambrogio, extends WSDL for the descriptions of QoS properties. A new extension could be defined for the specification of TT-WSDL, time-triggered WSDL. This could also be used to specify components using the TT-WSDL, so that the functionality and interface of components is described using the extension of an standardized language.

- SystemC extensions: The E-TTM meta-model implementation extends SystemC with the time-triggered MoC. The development of embedded systems requires heterogeneity of MoCs, so, the E-TTM should be used in conjuction with other extensions such as SystemC-AMS and HetSC.

We can't solve problems by using the same kind of thinking we used when we created them

ALBERT EINSTEIN

Appendix A

On fundamental attributes

This chapter analyses certain fundamental concepts of interest for embedded systems and their development such as complexity, dependability, composability, predictability, determinism, abstraction and consistency. As analyzed by the author, attributes such as determinism are inconsistently used within technical and scientific literature [PC07b], so, the purpose of this chapter is to analyze these fundamental attributes and identify their implicit requirements and implications.

A.1 Complexity

The nouns complexity and complication derive from the Latin word *complexus* and *complicare* that means to "fold together" [Oxf07]. Complication is "an involved or confused condition or state" [Oxf07] and complexity is the "the state or quality of being intricate or complicated" [Oxf07]. Simplicity is the antonym of complexity and simplification is the antonym of complication.

Definition: There are multiple computer-science domain specific definitions for the term complexity as shown in the following Table A.1:

Domain	Definition
General	"The degree to which a system or component has a design or implementation that is difficult to understand and verify" [dic91]
Mathematics	"The complexity of a process or algorithm is a measure of how difficult it is to perform. The study of the complexity of algorithms is known as complexity theory." [mat07]

continued on the next page

(*continued*)

Domain	Definition
Embedded Systems	"The number of system elements times the number of connections times the value for type of mutual reaction yields a measure for complexity." [Hei02, Rie00] "The complexity of a system relates to the number of parts, and the number and types of interactions among the parts, that must be considered to understand a particular function of the system" [Kop97]
Software	"Complexity is the extent of difficulty in programming" [YJ03] "Software complexity, the measure of how difficult the program is to comprehend and work with" [HMKD82]

Table A.1: Domain specific definitions for complexity.

Description: From a safety-critical embedded systems point of view, complexity is described as:

Complexity is quantifiable: expressed quantitatively using measurements and qualitatively (e.g., this solution is more complex).

Cognitive Complexity: The terms complexity and cognitive complexity are used interchangeably within this thesis. Cognitive complexity (see Section 2.1) of a given task describes the amount of cognitive resources required to perform a task, which is related to the amount of time required for the task and the amount of errors that occur [Rum06, Kop07, RE06]. The higher the cognitive complexity, the higher the amount of time and errors that can occur. Therefore it is of the utmost importance to reduce the cognitive complexity of the development of safety-critical embedded systems as stated by Jackson: "one key to achieving dependability at reasonable cost is a serious and sustained commitment to simplicity, including simplicity of critical functions and simplicity in system interactions" [JTM07].

Complexity source: As previously defined, complexity is "the degree to which a system or component has a design or implementation that is difficult to understand and verify" [dic91]. So, the real complexity source in the development of embedded systems is not the amount / type of functionalities and requirements to implement, but the lack of availability / usage of appropriate architectures, methodologies and tools that

might reduce the difficulty to understand, develop and verify embedded systems. What is complex today might not be complex tomorrow.

Computational complexity theory: deals with the classification of problems (P-problem, NP-problem and NP complete problem) based on the amounts of resources (e.g., execution time) required for the execution of algorithms.

Complexity management: is the simplification of a complex scenario in order to be processed by the limited cognitive capabilities of humans. Section 2.1.2 describes three basic simplification strategies: abstraction, partition and segmentation [Kop07, Rum06].

A.2 Dependability

Dependability is "the ability to avoid service failures that are more frequent and more severe than is acceptable" [ALRL04].

Definition: There are multiple definitions for the term dependability from which the most representatives are shown in the following Table A.2:

Domain	Definition
General	"trustworthy and reliable" [Oxf07]
Embedded Systems	"the ability to avoid service failures that are more frequent and more severe than is acceptable. It encompasses the following attributes: availability, reliability, safety, integrity and maintainability" [ALRL04] "A system is dependable when it can be depended on to produce the consequences for which it was designed, and no adverse effects, in its intended environment" [JTM07]

Table A.2: Dependability property definitions.

Description: The development of a dependable embedded-system that must provide a given service despite the occurrence of faults, requires the distribution of functions to achieve fault containment, error containment, fault tolerance and fault prevention [Kop97]. From a safety-critical embedded systems point of view, dependability involves and requires:

Dependability properties: Dependability encompasses the following prop-

erties [ALRL04].

- **Availability** is the "readiness for correct service" [ALRL04] that is usually expressed as "the probability that a system will be able to execute a function accurately at any given time" [dic00].
- **Reliability** is "the characteristic of an item or system expressed by the probability that it will perform a required mission under stated conditions for a stated mission time" [dic00].
- **Safety** is the "freedom from unacceptable risk" [iec98] an it is expressed as the probability that no critical failure will occur in a given interval of time $[0, t)$.
- **Integrity** is the "absence of improper system alterations" [ALRL04].
- **Maintainability** is the "ability to undergo modifications and repairs" [ALRL04] and alternatively "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" [dic00, dic91].
- **Security** is "a composite of the attributes of confidentiality, integrity, and availability" [ALRL04] that ensures "the protection of computer resources (e.g., hardware, software and data) from accidental or malicious access, use, modification, destruction or disclosure" [dic00].

Threats to dependability: According to Avizienis [ALRL04], the threats to dependability are failures, errors and faults. The error propagation follows a sequential propagation chain where faults can trigger errors, errors can trigger failures, failures can also trigger faults and so on.

- **Fault** is "the adjudged or hypothesized cause of an error" [ALRL04] which from the IEC-61508 point of view is defined as the "abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function" [iec98].
- **Error** is the "the part of the total state of the system that may lead to its subsequent service failure" [ALRL04] which from the IEC-61508 point of view is defined as the "discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition" [dic00, iec98].
- **Failure** is "an event that occurs when the delivered service deviates from correct service" [ALRL04] which from the IEC-61508 point of

view is defined as the "termination of the ability of a functional unit to perform the required function" [dic00, iec98].

Certification: Safety-critical embedded systems are often subject to certification, a formal assurance that the system has met technical standards to ensure the safety operation of the system. However, current "certification of the dependability of a software-based system usually relies more on assessments of the process used to develop the system than on the properties of the system itself" [JTM07].

The international standard IEC-61508 is the basic functional safety standard for many industry and transportation domains and it is supported by the DECOS architecture [SAS+07]. The standard covers the complete safety life cycle, covering both SW and HW aspects, and it has been the baseline or reference to develop sector specific standards such as: railway EN-5102X [en501, en503], vertical transportation (lift) EN-81-1/prA1 [en804], automotive ISO-WD-26262, process industry sector (e.g., oil industry) IEC-61511 [std04], machinery sector IEC-62061 [std05], nuclear plants IEC-61513 [std01], medical IEC-60601 [iec], etc.

Dependability as a property of the complete system: Dependability is not a local property of modules or subsystems (e.g., software module) but must be articulated and evaluated from a systems perspective that takes into account the usage context. In addition to this, the consequences and intended environment must be stated explicitly with a clear priorization of the system requirements and environmental assumptions [JTM07].

Dependability and complexity are strongly linked where "one key to achieving dependability at reasonable cost is a serious and sustained commitment to simplicity, including simplicity of critical functions and simplicity in system interactions" [JTM07]. In addition to safety, "the link between complexity and security is a well-accepted fact in system security engineering" [LT07] based on the security design principles outlined by Schroeder and Saltzer [SS75]. The principle of psychological acceptability states that the introduction of a security mechanism should not make the system more complex than it is without it. The principle of economy of mechanisms recommends that system security mechanisms should be kept as simple as possible [LT07].

The three Es: A recent report by the 'National Academy of Sciences' of software dependability stated that in order to develop dependable systems cost effectively three key points, the three Es, must be considered [JTM07]:

Explicit claims: "No system can be dependable in all respects and under all conditions. So to be useful, a claim of dependability must be explicit." [JTM07]

Evidence: Dependability claims require concrete evidences that substantiates the dependability claim, which will take the form of a 'dependability case'.

Expertise: in dependable embedded systems and the application domain under consideration is required to achieve dependable systems.

A.3 Composability

The nouns composability and composition derive from the Latin word *componere* that means to "put together" [Oxf07] and are generally defined as:

- Composition: "The act of combining parts or elements to form a whole" [Web89, KS03, Kop04, SKMVM04] and "the action of putting things together; formation or construction" [Oxf07].
- Composability: "The ease of forming a whole by combining parts" [Web89, KS03, Kop04, SKMVM04, Rum06].

Definition: There are multiple computer-science domain specific definitions for the terms composition and composability, as listed in the following table, from which the definition made by Kopetz [Kop97] has been used in this thesis.

Domain	Definition
Embedded Systems	"Composability: Ability to link subsystems so that properties established at subsystem levels hold at the system level" [Szt00, SS01] "An architecture is said to be composable with respect to a specified property if the system integration will not invalidate this property, once the property has been established at the subsystem level. Examples of such properties are timeliness or testability. In a composable architecture, the system properties follow from the subsystem properties" [Kop97]
Software and OOP	"Composability is a property of a software component meaning that it may easily and systematically be combined with other components" [Bar02] "Composability allows for the modular specification of modules with multiple independent concerns" [Ber96, WRMS03]

continued on the next page

(*continued*)

Domain	Definition
Modelling	"Composability is the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements" [PW03a] "Model composability is concerned with techniques for developing a whole model of a system from the models of its sub-systems." [HD05]

Table A.3: Domain specific definitions for composition and composability.

Description: From a safety-critical embedded systems point of view, composability involves the following considerations:

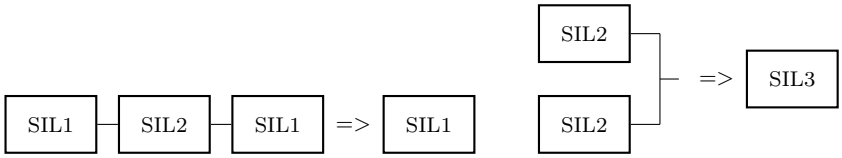
Composability is a boolean property: Composability requires that system integration will not invalidate properties once the properties have been established at subsystem level. Therefore composability is a boolean property, an item is either composable (does not invalidate properties) or not composable (invalidates properties) [Kop04].

It requires defining which attribute and under which conditions: Composability always refers to a given attribute and set of conditions / scenarios. Therefore, the definition of what composability is an attribute of and under which conditions / scenarios (if any) is required.

Composability and interoperability are not the same: The adjective interoperable is defined as "able to operate in conjunction" [Oxf07] and the adjective compatible as "able to exist or be used together without problems or conflict" [Oxf07]. Functional composability goes beyond functional interoperability / compatibility, which are necessary but not sufficient to ensure that properties are not invalidated during integration.

Functional composability (syntactic and semantic): Functional composability is the capability to select, combine and assemble components. Two functional composability types are distinguished: syntactic composability determines whether the components be connected [ST07] while semantic composability is a question of whether the composed components can be meaningfully composed [PW03b].

Safety composability: "So, is safety... composable per se?" [Sur06]. Safety might be a composable property if required architectural and certification safety constraints are met. For example, as shown in Figure A.1, the IEC-61508-2 [IEC00] describes the composition rules of given serial and parallel subsystems to produce aggregate systems of known Safety Integrity Level (SIL).



(a) serial (b) parallel

Figure A.1: IEC-61508 safety composition for serial and parallel channels.

Architectural composability: An architecture that supports composability (e.g., time domain) ensures that services established at the component level will also function properly at the system level and that the integration of the components proceeds without unintended side effects [KS03]. For an architecture to be composable (functional and time domain) Kopetz has specified that it must comply with the following four principles with respect to the RS interfaces [Kop00a, KS03, Kop04]: independent development of components, stability of prior services, constructive integration of components and replica determinism behaviour.

Composability requirements: Different types of embedded systems require different composability requirements as shown in Table A.4.

Type	**Composability of properties**				
	Functional	**Distributed**	**Time Domain**	**Value Domain**	**Safety**
Generic	Yes	Optional	-	-	-
Real-Time	Yes	Optional	Yes	-	-
Safety Critical	Yes	Yes	Yes	Yes	Yes

Table A.4: Composability requirements for embedded systems.

Algebraic notation

Composition can also be algebraically represented. First, composition of generic services is algebraically represented and then this is applied to represent the composition of jobs and DASes [wwwc, KOPS04].

Composition of Services: It is possible to define the composition scheme ($\oplus$) of services (S) as a function where the dimension (D) corresponds to the number of properties / attributes that are composable (e.g., time).

$$\oplus : S^D \oplus S^D \rightarrow S^D \tag{A.1}$$

Basic Service: Each service is defined by a Linking Interface (LIF) that defines the service behaviour in all composable dimensions D (e.g., time and value domain) and which must meet a set of architectural constraints ($\{A_c\}$). The definition of the LIF is required by the first composability principle, independent development of components, while the architectural constraints define the constraints required to meet the third principle, constructive integration. Each single service is the composition of the main intended service ($S_{i,0}$) plus additional sigma potential additional emerging services ($\delta_{i,j}$).

$$\begin{gathered} LIF_i^D \in \{A_c\} \Rightarrow S_i \\ S_i = S_{i,0} \oplus \sum_{j=0}^{j=\infty} \delta_{S_{i,j}} = S_{i,0} \oplus \delta_{S_{i,0}} \oplus \delta_{S_{i,1}} \oplus \delta_{S_{i,2}} \oplus ... \end{gathered} \tag{A.2}$$

Composition of services: The composition of N services is defined as follows and meets the second principle, stability of prior services, where the composition of services does not change previous single services but could bring new emerging services [SKMVM04].

$$\begin{gathered} \{S\}_N = \sum_{i=1}^{i=N} S_i = S_0 \oplus S_1 \oplus ... \oplus S_{N-1} \\ \{S\}_N = \sum_{i=1}^{i=N} S_i = \sum_{i=1}^{i=N} \left(S_{i,0} \oplus \sum_{j=0}^{j=\infty} \delta_{S_{i,j}} \right) \Rightarrow \{S\}_N \geq \sum_{i=1}^{i=N} (S_{i,0}) \end{gathered} \tag{A.3}$$

In a similar way, the composition of services is a composition of LIF that must meet a set of architectural constraints:

$$\{S\}_N = \sum_{i=1}^{i=N} S_i = \left(\sum_{i=1}^{i=N} \left(LIF_i^D \right) \right) \in \{A_c\} \rightarrow \left\{ LIF^D \right\}_N \in \{A_c\} \tag{A.4}$$

Composition of properties: By definition, composability is an attribute of a set of properties ($\{P\}_D$) where the composition of services does not invalidate the properties of single services. So, every service (S_i) has a

set of properties that are not invalidated by the composition operator as long as services meet the architectural constraints.

$$S_i \rightarrow \{p\}_i$$
$$\{S\}_N = \sum_{i=1}^{i=N} \{p\}_i = \{p_1\}_D \oplus \{p_2\}_D \oplus ... \oplus \{p_N\}_D = \{\{p\}_D\}_N \tag{A.5}$$

Composition of DAS and jobs: Following the given algebraic representation a job (j) is a basic service, DAS is a composition of jobs and system application is a composition of DASes:

$$j_i = j_{i,0} \oplus \sum_{j=0}^{j=\infty} \delta_j = j_{i,0} \oplus j_{i,1} \oplus ...$$
$$DAS = \{j\}_N = \sum_{i=1}^{i=N} j_i = j_0 \oplus j_1 \oplus ... \oplus j_{N-1} \tag{A.6}$$
$$app = \{DAS\}_M = \sum_{i=0}^{j=M} DAS_i$$

A.4 Predictability

The verb to predict derives from the Latin word *praedict* that means to "made known beforehand, declared" [Oxf07] and it is generally defined as to "say or estimate that (a specified thing) will happen in the future or will be a consequence of something" [Oxf07].

Definition: There are multiple definitions for the noun predictability, some of them domain specific, as listed in the following table:

Domain	Definition
Software	"... the ability of the software firm to accurately estimate the needed resources, time, performance, quality and functionality of its software projects" [AFC03]
Scheduling Alg.	"A scheduling algorithm is predictable if and only if for any set of jobs the finish time for the actual execution time is no later than the finish time for the maximum execution time" [RL94, XSH+06]
MBD	"Predictability in the component-based development approach is the ability to reason about application behaviour from the quality attributes of the components and the components interconnections." [Lar04]

continued on the next page

(*continued*)

Domain	Definition
RT Embedded	"One of the most important properties that a hard real-time system should have is predictability [SR90]. That is, based on the kernel features and on the information associated with each task, the system should be able to predict the evolution of the tasks and guarantee in advance that all critical timing constraints will be met" [But04]

Table A.5: Domain specific definitions for predictable and predictability.

Description: From a safety-critical embedded-systems point of view, predictability involves the following considerations:

Predictability is quantifiable To predict is defined as "say or estimate that (a specified thing) will happen in the future or will be a consequence of something" [Oxf07] and predictability refers to the degree that an output or attribute meets the expected value. This degree is quantifiable and can be expressed quantitatively (e.g., there is 80% probability) and qualitatively (e.g., this solution provides more predictability).

Definition of what and under which conditions The definition of which attribute is predictable and under which conditions and scenarios (if any) is required. For example an architecture might have multiple attributes, and therefore which one is the predictable attribute and under which conditions has to be defined, e.g., time predictability under a single fault-hypothesis.

Predictability, entropy and determinism Predictability is a quantifiable property, where null predictability is entropy and full predictability with no stochastic argument is determinism.

Predictable and potentially predictable Predictable attributes that are not feasible to predict due to resource limitations such as computational effort and time could be called potentially predictable. For example, "For the calculation of the Worst-Case Execution Time of a program, the ideal thing would be to measure (or simulate) all the possible execution paths. This is generally not affordable because it would be very expensive in time." [RS03]. So, even though the WCET calculation is predictable, in reality it might just be potentially predictable due to resource limitations.

Time predictability is required for safety-critical embedded systems in order to ensure that timing constraints are met [XP90, HLTW03, ZLL04, WT06]. Therefore, the timing behaviour of real-time tasks is usually assessed based on WCET analysis [WKPR05]. In static real-time systems "if the prediction is that 100% of all tasks over the entire life of the system will meet their deadlines, then the system is predictable without resorting to any stochastic evaluation. In dynamic real-time systems we must resort to stochastic evaluation for part of the performance evaluation" [SR90].

Threats to time domain predictability Threats to time predictability of embedded systems can be found for example in the processor architecture, software, network, operating system and scheduling dependencies on several levels [TW04, ZLL04, WT06]

Processor architecture Processor architectures are usually optimized for average-case performance and not for predictable performance [HLTW03, ZLL04, WKPR05, WT06]. Modern microprocessors are composed of thousands of millions of transistors that interact among them and define functional blocks such as execution pipelines, memory caches, execution units, etc. that aim to improve average performance but which are not designed to be predictable in the time domain.

For example caches add unpredictability to the execution timing analysis because "the timing behaviour of an individual instruction cannot be determined locally, but depends on the execution history" [TW04] and this can increase the variability of execution times [But04, WT06]. The addition of multiple hardware mechanisms in order to improve the average performance of modern processors has not only made them non time deterministic, but they can also be considered to be chaotic [BGPT06]. This unpredictability of the internal microprocessor state has even also been used to define pseudorandom number generators for cryptographic purposes [SS03].

Software Due to limited human cognitive capabilities, "nontrivial software written with threads, semaphores and mutexes is incomprehensible to humans" [Lee06]. In general, it is difficult or even impossible to analyze and validate the schedulability of real-time systems, ensuring timeliness of real time systems, due to the various forms of dependencies among tasks such as shared resources, priority inversion and multiple kinds of conflicts and rare event situations [ZLL04, Lee06]. In addition to this, even the programming language might be a source of unpredictability due to the lack of sup-

port in the definition of explicit time constraints on task execution (e.g., Ada, C), non-deterministic constructs (e.g., select statement in Ada), etc. [But04]

***Real-Time Operating Systems* (RTOS)** must provide multiple functionalities and manage multiple resources that can be sources of unpredictability in the real-time system: thread scheduling and synchronization, concurrency control protocol for accessing shared resources, kernel preemptability, timer resolution, network protocol stack, etc. [But04, ZLL04]. The use of virtual resources such as virtual memory paging makes timing analysis even more difficult [PH07].

Scheduling on several levels Distributed safety-critical applications have static and / or dynamic scheduling on several levels that might lead to unpredictable behaviour [TW04] at a system level. Example scheduling levels are:

- System application level: local vs. distributed task scheduling of computation and communication
- Node application level: task scheduling, multi-threading scheduling and share resources scheduling
- Microprocessor level: static vs. dynamic instruction scheduling

In order to provide time predictability in a complex hard real-time embedded system, major task (timing) characteristics must be known or bounded in advance because otherwise it would be impossible to guarantee a priori that all timing constraints are met [XP93].

Threats to value domain predictability Almost every microprocessor as well as many programming languages from C to Java have specified the floating-point arithmetic to be IEEE-754 [iee85] compliant [Sch03]. However, IEEE-754 standardizes a few basic operations, and functions such as trigonometric sine are not specified by this standard. Therefore, the results obtained by executing the same mathematical application code in different targets depend on the combination of the microprocessor, compiler, libraries and runtime environment. These discrepancies, also known as the 'consistent comparison problem' [BBKL89], might not be negligible and could even have noticeable consequences: they could lead to value domain unpredictability [Par01, Mon07].

Threats to functional predictability System updates and system upgrades are two important sources of behaviour unpredictability (time and value domain) because "Even small changes can result

in unexpected and difficult to resolve failures. Eventually, these changes exceed the capacity of the system" [FLV00]. It is difficult to ensure that behavioural constraints defined in the specification and design are guaranteed in the execution platform [FLV00] and this requires sound methods and tools to derive reliable and precise run-time guarantees [TW04]. Testing provides a limited guarantee only [Car], because the testing notion of predictability relies on the premise that if we observe enough past executions we can predict future behaviour and this is not a sound method.

Layer by layer predictability It has been stated that in order to ensure predictability, the complete system development must be predictable from the model to the system architecture, language, operating system, etc. That means that a layer by layer predictability is required [SR90, CVH93, TW04].

A.5 Determinism

The noun determinism derives from the Latin word *determinare*, which means to "limit, fix" [Oxf07] and it is generally defined as "Pertaining to a process, model or variable whose outcome, result, or value does not depend on chance. Contrast: stochastic" [dic91, dic00].

Definition: There are other multiple definitions for the adjective deterministic, some of them domain specific, as listed in Table A.6.

Domain	Definition
Mathematics	"Stochastic is often used as counterpart of the word deterministic, which means that random phenomena are not involved. Therefore, stochastic models are based on random trials, while deterministic models always produce the same output for a given starting condition." [mat07]
Replica Determinism	"A set of node is replicate determinate, if all the nodes in this set contain the same externally visible h-state at their ground state, and produce the same output messages at points in time that are at most an interval of time units apart" [Kop97] "... fault-free replicated components are required to exhibit replica determinism i.e., they have to deliver identical outputs in an identical order within a specified time interval" [PBWB00]

continued on the next page

(*continued*)

Domain	Definition
Model	"A model behaves deterministically if and only if, given a full set of initial conditions (the initial state) at time T0, and a sequence of future inputs, the outputs at any future instant T are entailed" [Kop07] in the context of sparse time base.
Software	"If a software component is called twice with the same input values at the same time instants, it should both times produce the same output values at the same time instants." [PT05]
Communication	"Two correctly operating independent deterministic communication channels will deliver messages always in the same order." [Kop06]
System	"the outputs of the system should be uniquely determined by its inputs and possibly by their timing" [Ber00]

Table A.6: Domain specific definitions for determinism and deterministic.

Description: All in all, determinism means that for a given set of relevant conditions a given item (e.g., property, output, etc.) is completely predictable and does not depend on randomness or stochastic statements. This implies that an output (o) is deterministic for a given set of relevant conditions (c), if given the same set of initial conditions then the system (s) always generates the same outputs at the same sparse-time (t) when given the same inputs (i) at the same sparse-time (ts), as shown in Equation A.7. The definition and description of sparse-time concept can be found in [Kop97].

$$\begin{aligned} \forall i, o, o' \in c\,(< s, i, ts_i > \rightarrow < o, ts_o > \wedge < s, i, ts_i > \rightarrow < o', ts_{o'} >) \\ \Rightarrow (o = o'; ts_o = ts_{o'}) \end{aligned} \tag{A.7}$$

The determinism concept involves the following considerations:

Determinism is a boolean property: an item is either deterministic or not because the algebraic Equation A.7 must be met for all inputs and outputs under the given set of conditions.

Requires defining what and under which conditions: The terms determinism and deterministic require the definition of 'what' attribute and under 'which' conditions: 'what' is deterministic for 'which' conditions.

No randomness involved: Determinism by definition should not rely on random or stochastic arguments, thus references to stochastic arguments

should be avoided. In addition to this, the proof of determinism of a given attribute should not be based on non-exhaustive scenario verification and simulation because they could miss rare events [CSEF06].

Stochastic attributes cannot be deterministic (e.g., dependability): Determinism property should not be applied to items that are intrinsically stochastic. As an example dependability cannot be deterministic because dependability is interpreted in a probabilistic sense: "The extent to which a system possesses the attributes of dependability should be interpreted in a relative, probabilistic sense, and not in an absolute, deterministic sense: due to the unavoidable presence or occurrence of faults, systems are never totally available, reliable, safe, or secure" [ALR00]. However, dependability can be quantified by deterministic or probabilistic measures [Nel90].

Determinism and predictability are not the same: Determinism is a boolean property while predictability is quantifiable, null predictability is entropy and full predictability with no stochastic argument is determinism. In addition to this, deterministic rules and models imply that the outcome is fully predictable, yet the computational effort required to predict the outcome might sometimes limit the real feasibility (potentially predictable): "We cannot determine the behaviour of the system not because we cannot 'know how it works', but because its complexity exceeds our computing or perceptual capabilities" [Ger02].

Determinism and composability: The development of embedded systems based on composable architectures (e.g., TTA [Kop00a]) benefits from the capability to reason about the properties of the composed system based on the architectural composable properties and not in the specific implementation and integration of components [Kop04, PC07a]. If an architecture is not composable for a given property (e.g., time) it cannot be deterministic for that property, because the lack of composability means that it is not possible to predict the properties of all possible compositions without stochastic arguments. Architectural instances (systems) will not inherit property determinism from the architecture but specific instances could be designed and proved to be deterministic based on a costly implementation specific analysis.

Determinism and simultaneity: Determinism in the time domain must deal with the simultaneity of events that must be established consistently within the distributed embedded system. The sparse time model is required to handle simultaneity in the absence of agreement protocol, because "it is principally impossible to always arrive at a system-wide

consistent notion of simultaneity in a distributed system that allows the occurrence of physically distributed events at any instant of the dense timeline" [Kop07].

Abstraction Level, building determinism from non-determinism
Multiple safety-critical embedded systems are defined as deterministic in the value and time domain under a single fault hypothesis. Determinism in the time and value domain is a system property required by safety-critical embedded systems, which implicitly requires every safety-critical sub-system to be also deterministic in both domains. However, at a lower abstraction level systems are made of components that are not deterministic under a single fault hypothesis (e.g., single communication bus) but at the system abstraction level the non-determinism of the single fault hypothesis is masked (e.g., redundant communication bus). Thus, under some circumstances determinism can be built from non-determinism.

A.6 Abstraction

The noun abstraction derives from the Latin verb *abstrahere* that means to "draw away" [Oxf07] and it is generally defined as " the process of considering something independently of its associations or attributes" [Oxf07].

Definition: There are multiple computer-science domain specific definitions for the term abstraction as listed in the following Table A.7:

Domain	Definition
Embedded System	"a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information" [dic91]
Model	" Abstraction is the activity that tries to remove (or hide) irrelevant information, which improves the comprehensibility of existing design models and facilitates the evaluation of different design solutions." [HVF+05]

Table A.7: Domain specific definitions for abstraction.

Description: Raising the level of abstraction is key towards the development of small and large dependable embedded systems due to our limited cognitive

capabilities for understanding the overall system [Kop03, SLMR05]. From an embedded systems development point of view abstraction involves and requires the following:

Safety-critical embedded systems: For safety-critical embedded systems 'the information relevant to a particular purpose' is dependability, functional composability and determinism in the value and time domain properties. So, abstraction in this domain is achieved by 'ignoring the remainder of the information'.

Abstraction and models: Models provide a "higher abstraction level than code, thus, they are less connected to their target platforms" [SLMR05]. Therefore, the specification and design of embedded systems with suitable models that do not deal with software and hardware details (e.g., software classes) but with dependability and time and value domain correctness would provide a higher abstraction level.

Current abstraction limitations: Even though the hardware industry has kept pace with Moore's law during last 25 years, software development and programming languages seem not to have changed at this pace. In fact, embedded software engineers use the same programming languages as 20-30 years ago (e.g., C, Ada and ASM). C is the most widely used programming language that even though it was designed in the early 1970s when most applications ran on a single processor with few resources (e.g., memory), nowadays it is widely used to program distributed safety-critical multi-processor systems. In addition to this, "programming language semantics do not handle time, so developers can only specify timing requirements indirectly" [Lee05]. This is also the case for UML modeling language which right now does not specify the time domain with the required rigour [Kop00b].

A.7 Consistency

The noun consistency derives from the Latin word *consistentia* that means to "standing firm or still, existing" [Oxf07] and the adjective consistent is generally defined as "acting or done in the same way over time, especially so as to be fair or accurate; not containing any logical contradictions" [Oxf07].

Definition: There are other multiple definitions for the adjective consistent and noun consistency, some of them domain specific, as described in Table A.8. The term consistency used within this thesis will refer to the definition

made by Besana [BB03] and will usually be used in the context of safety-critical embedded systems development.

Domain	Definition
Embedded System	"Consistency is the property of maintaining the same behaviour at different levels of abstraction through synthesis and refinement, leading to functionally correct implementation." [BB03]

Table A.8: Domain specific definitions for consistent and consistency.

Description: The consistency concept involves and requires the following considerations from a safety-critical embedded systems development point of view:

Consistency is a boolean property: an item is either consistent or not because properties and constraints of interest are either preserved or not preserved.

Requires defining what and under which conditions: The terms consistent and consistency require the definition of 'what' properties / constraints and under 'which' conditions: 'what' is consistent for 'which' conditions / scenarios.

Consistency and composability: Composability, ensuring that system integration will not invalidate the properties of subsystems, can be seen as the mechanism required to ensure development consistency. That means, development consistency requires development composability of selected properties and constraints for a given development process. In addition to this, consistency is also required to develop composable systems in which selected properties and constraints are preserved. Thus, both consistency and composable are bidirectionally linked.

Consistency and determinism: Determinism, predictability with no stochastic argument, can be seen as the basic property of any given development process that must ensure consistency because it must be able to always predict that selected properties and constraints are preserved throughout the development process if certain rules are met. In addition to this, consistency is also required to develop deterministic systems in which selected properties and constraints are preserved. Thus, both consistency and determinism are bidirectionally linked.

Consistency and dependability: The development of safety-critical embedded systems requires the highest levels of dependability, in which testing alone will rarely suffice and will have to be augmented by analysis. This analysis, involves a rigorous process which will be needed to ensure that the chain of evidence for dependability claims is preserved [JTM07]. Thus, consistency of the development process ensuring that selected properties and constraints are preserved during the development is key towards the development of cost-effective highly dependable embedded systems.

Appendix B

List of Acronyms

Ada Ada programming language

ADC Analog Digital Converter

AI Artificial Intelligence

AMS Analog Mixed Signal

ASM Assembly Language

ATP Automatic Train Protection

AUTOSAR AUTomotive Open System Architecture

BTM Balise Transmission Module

C C Programming Language

C++ C++ Programming Language

CC Communication Controller

CCSL Clocked Constraint Specification Language

CNI Communication Network Interface

COTS Commercial-Off-The-Self

CP Configuration and Planning

CPU Central Processing Unit

CT Continuous Time

DAS Distributed Application Subsystem

DECOS Dependable Embedded Components and Systems

DM Diagnostic and Maintenance

DMI Driver Machine Interface

DT Discrete Time

ECU Electronic Control Unit

E/E Electrics / Electronics

ERTMS European Rail Traffic Management System

ETCS European Train Control System

EVC European Vital Computer

E-TTM Executable Time-Triggered Model

FMEA Failure Mode and Effect Analysis

GNU GNU's Not Unix!

FSM Finite State Machine

HDL Hardware Description Language

HetSC Heterogeneous Specifications in SystemC

HMI Human Machine Interface

HW Hardware

IEC International Electrotechnical Commission

IEC-61131 IEC standard for PLCs

IEC-61508 IEC standard for "Functional safety of electrical / electronic / programmable electronic safety-related systems (E/E/PES)"

IEC-61499 IEC open standard for distributed control and automation

IEC-61588 IEC precision clock synchronization protocol for networked measurement and control systems

IEEE Institute of Electrical and Electronics Engineers

IEEE-1588 IEEE precision clock synchronization protocol for networked measurement and control systems

IEEE-1666 IEEE Standard SystemC Language Reference Manual

IEEE-754 IEEE Standard for Binary Floating-Point Arithmetic

IETF Internet Engineering Task Force

I/F Interface

IGCT Integrated Gate Commutated Thyristor

INRIA Institut National de Recherche en Informatique et en Automatique

IP Intellectual Property

ISBN International Standard Book Number

ISO International Organization for Standardization

GPS Global Positioning System

GSM/R Global System for Mobile Communications - Railway

JRU Juridical Recorder Unit

KB Kilobyte

LET Logical Execution Time

LIF Linking Interface

LOC Lines Of Code

LRMB Layered Reference Model of the Brain

LTM Loop Transmission Module

MARTE Modeling and Analysis of Real-Time Embedded Systems

Matlab Matrix Laboratory; Numerical computing environment and programming language created by Mathworks

MEDL Message Descriptor List

MDA Model Driven Architecture

MBD Model-Based Design

mmE-TTM E-TTM meta-model

MoC Model of Computation

MVB Multifunction Vehicle Bus

N/A Not Applicable

NID Namespace Identifier

NoC Network-on-Chip

NSS Namespace Specific String

NTP Network Time Protocol

ODS Object Data Store

OMG Object Management Group

OO Object-Oriented

OOP Object-Oriented Programming

PFSM Periodic Finite State Machine

PIM Platform Independent Model

PLC Programmable Logic Controllers

PSM Platform Specific Model

PTP Precision Time Protocol, see IEEE-1588

QoS Quality of Service

RN Resource Name

RS Real-Time Service

RT Real-Time

RTE Run Time Environment

RTES Real-Time Embedded System

RTOS Real-Time Operating Systems

SCADE Safety Critical Application Development Environment

SCES Safety-Critical Embedded-Systems

SDF Synchronous Data Flow

SDL System Description Language

SFI Simulated Fault Injection

SI International System of Units

SIL Safety Integrity Level

SLDL System-Level Design Language

SW Software

SysML Systems Modelling Language

SystemC SystemC SDL / SLDL

SystemC-AMS SystemC - Analog Mixed Signal

TAI International Atomic Time

TCP/IP Transmission Control Protocol (TCP) / Internet Protocol (IP)

TDL Timing Definition Language

TIMMO Timing Model

TIU Train Interface Unit

TLM Transaction-Level Model

TMO Time-triggered Message-triggered Object

TMR Triple Modular Redundancy

TT Time-Triggered

TTA Time-Triggered Architecture

TTE Time-Triggered Ethernet

TTNoC Time-Triggered Network-on-Chip

TTP Time-Triggered Protocol

TTP/A TTP Class A

TTP/C TTP Class C

TTTech Time-Triggered Technologies

UI User Interface

UML Unified Modelling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

UTC Coordinated Universal Time

UTF Uniform Time Format

VHDL VHSIC HDL

VHSIC Very-High-Speed Integrated Circuits

Verilog Verilog is an HDL

VFB Virtual Functional Bus

V/F Voltage / Frequency

WCCOM Worst-Case Communication delay

WCET Worst-Case Execution Time

Winsock Windows Sockets API

WS Web Service

WSDL Web Service Description Language

xfE-TTM E-TTM execution framework

Bibliography

[AAH97] David W. Allan, Neil Ashby, and Clifford C. Hodge. The science of timekeeping. Technical Report APP1289, Agilent Technologies, 1997.

[AFC03] Stephano Ah-Fock and Angele Cavaye. The effect of reusability on perceived competitive performance of australian software firms. *Journal of Research and Practice in Information Technology*, 35(3):183–196, 2003.

[All86] Jams F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(1):832–843, 1986.

[ALR00] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. In *Third Information Survivability Workshop (ISW)*, Boston, USA, 2000.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. In *IEEE Transactions on Dependable and Secure Computing*, volume 1, pages 11–33, 2004.

[AMdS07] Charles Andre, Frederic Mallet, and Robert de Simone. Time modeling in MARTE. In *Forum on Specification & Design Languages (FSDL)*, Barcelona, 2007.

[AMPF07] Charles Andre, Frederic Mallet, and M-A Peraldi-Frati. A multiform time approach to real-time system modeling. In *International Symposium on Industrial Embedded Systems (SIES)*, pages 234–241, Lisbon, 2007.

[AUT05] AUTOSAR. AUTOSAR press information pack, 6-7 October 2005.

[aut06] Applying Simulink to AUTOSAR. Technical report, AUTOSAR, 2006.

[Bar02] F. Barbier. Composability for software components: an approach based on the whole-part theory. In *Eighth IEEE International Conference on Engineering of Complex Computer Systems*, pages 101–106, 2002.

[BB03] M. Besana and M. Borgatti. Application mapping to a hardware platform through automated code generation targeting a RTOS: a design case study. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 41–44, 2003.

[BBKL89] S. S. Brilliant, S. S. Brilliant, J. C. Knight, and N. G. Leveson. The consistent comparison problem in N-version software. *Transactions on Software Engineering*, 15(11):1481–1485, 1989.

[BCE+03] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[Ber96] Lodewijk M.J. Bergmans. Composability: Why, what, and how? In *In Workshop on Composability Issues in Object-Orientation, Tenth European Conference on Object-Oriented Programming*, Linz, Austria, 1996.

[Ber00] G. Berry. The foundations of Esterel. *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 2000.

[Ber03] G. Berry. The effectiveness of synchronous languages for the development of safety-critical systems. page 15. Esterel Technologies, 2003.

[BFLS01] Andrea Bondavalli, Alessandro Fantechi, Diego Latella, and Luca Simoncini. Design validation of embedded dependable systems. *IEEE Micro*, 21(5):52–62, 2001.

[BGPT06] Hugues Berry, Daniel Gracia Perez, and Olivier Temam. Chaos in computer performance. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, (16):15, 2006.

[BIP08] BIPM. The international system of units (SI) - 8th edition, 2008.

[BL05] Tim Berners-Lee. RFC 3986 - unifirm resource identifier. Technical report, The Internet Engineering Task Force (IETF), 2005.

[BM05] Catalin Buhusi and Warren H. Meck. What makes us tick? functional and neural mechanisms of interval timing. *Nature Reviews Neuroscience*, 6(10):755–765, 2005.

[BMS08] Cristiana Bolchini, Antonio Miele, and Donatella Sciuto. Fault models and injection strategies in systemc specifications. In *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD)*, pages 88–95, 2008.

[BMW93] T. J. Biggerstaff, B. G. Mitbander, and D. Webster. The concept assignment problem in program understanding. In *15th International Conference on Software Engineering*, pages 482–498, 1993.

[Bru72] B. C. Bruce. A model for temporal references and its application in a question answering program. *Artificial Intelligence*, 3:1–25, 1972.

[BS07] Jens Brandt and Klasus Schneider. How different are Esterel and SystemC. In Eugenio Villar, editor, *Forum on Design Languages (FDL)*, Barcelona, 2007. Springer.

[But04] Giorgio C. Buttazzo. *Hard real-time computing systems: Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series. Springer, 2 edition, 2004.

[Car] CarnegieMellon. Predictability by construction - building high-stakes systems from certified software components. Technical report, Carnegie Mellon.

[CCM+03] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, and Peter Niebert. From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications, 2003.

[Cow01] Nelson Cowan. The magical number 4 in short-term memory: a reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, 2001.

[CSEF06] Hussein Charara, Jean-Luc Scharbarg, J. Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *18th Euromicro Conference on Real-Time Systems*, page 10, 2006.

[CVH93] Matjaz Colnaric, Domen Verber, and Wolfgang A. Halang. Design of embedded hard real-time applications with predictable behaviour. In *Real-Time Applications, 1993., Proceedings of the IEEE Workshop on*, pages 193–197, 1993.

[D'A06] Andrea D'Ambrogio. A model-driven WSDL extension for describing the QoS of web services. In *International Conference on Web Services (ICWS '06)*, pages 789–796, 2006.

[Dav95] Paul Davies. *About Time: Einstein's Unfinished Revolution*. Simon & Schuster paperbacks, 1995.

[dic91] *IEEE Standard Computer Dictionary - Compilation of IEEE Standard Computer Glossaries*. IEEE, New York, 1991.

[dic00] *The Authoritative Dictionary of IEEE Standards Terms*. Standards Information Network, IEEE Press, New York, 7th edition, 2000.

[DLSMG04] Bernard Dion, Thierry Le Sergent, Bruno Martin, and Herbert Griebel. Model-based development for time-triggered architectures. In *The 23rd Digital Avionics Systems Conference (DASC)*, volume 2, page 6, 2004.

[DP05] F. Deissenbock and M. Pizka. Concise and consistent naming [software system identifier naming]. In *13th International Workshop on Program Comprehension (IWPC)*, pages 97–106, 2005.

[Eag08] David M. Eagleman. Human time perception and its illusions. *Current Opinion in Neurobiology*, 18(2):131–136, 2008.

[Edw99] S. A. Edwards. Compiling Esterel into sequential code. In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign (CODES)*, pages 147–151, 1999.

[EJ00] Cecilia Ekelin and Jan Jonsson. Solving embedded system scheduling problems using constraint programming. Technical report, 2000.

[en501] EN50128:1997 - railway applications: Software for railway control and protection systems, 2001.

[en503] EN50129:2003 - railway applications. communication, signalling and processing systems. safety related electronic systems for signalling, 2003.

[en804] EN81-1:1998/prA1:2004 - safety rules for the construction and installation of lifts part 1: Electric lifts, 2004-02 2004.

[EZ07] A. Lee Edwards and Haiyang Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *Proceedings of the 7th ACM and IEEE international conference on Embedded software*, pages 114–123, Salzburg, Austria, 2007.

[FDNG+09] Alberto Ferrari, Marco Di Natale, Giacomo Gentile, Giovanni Reggiani, and Paolo Gai. Time and memory tradeoffs in the implementation of AUTOSAR components. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 864–869, 2009.

[FHPS04] Claudiu Farcas, Michael Holzmann, H. Pletzer, and G. Stieglbauer. The TDL advantage. Technical report, 2004.

[FLV00] Peter H. Feiler, Bruce Lewis, and Steve Vestal. Improving predictability in embedded real-time systems. Technical Report CMU/SEI-2000-SR-011, Carnegie Mellon, December 2000.

[Gal03] Peter Galison. *Einstein's Clocks, Poincare's Maps: Empires of Time*. W. W. Norton & Company, 2003.

[Ger02] Carlos Gershenson. Complex philosophy. In *Proceedings of the 1st Biennial Seminar on Philosophical, Methodological & Epistemological Implications of Complexity Theory*, La Habana, Cuba, 2002.

[GG06] Sumit Ghosh and Norbert Giambiasi. Modeling and simulation of mixed-signal electronic designs - enabling analog and discrete subsystems to be represented uniformly within a single framework. *IEEE Circuits and Devices Magazine*, 22(6):47–52, 2006.

[Gho99] Sumit K. Ghosh. *Hardware Description Languages: Concepts and Principles*. Wiley-IEEE Press, 1999.

[Gho02] S. Ghosh. In search of the origin of VHDL's delta delays. In *International Symposium on Quality Electronic Design*, pages 310–315, 2002.

[GLMS02] Thorsten Grotker, Stan Liao, Grant Martin, and Stuart Swan. *System design with SystemC*. Kluwer Academic Publishers Group, 2002.

[HA04] U. Hatnik and S. Altmann. Using ModelSim, Matlab/Simulink and NS for simulation of distributed systems. In *International Conference on Parallel Computing in Electrical Engineering (PARELEC)*, pages 114–119, 2004.

[Hal98] Nicolas Halbwachs. Synchronous programming of reactive systems, 1998.

[Ham07] Christoph Hammerschmidt. AUTOSAR standard not ready to plug-and-play. *EETimes Europe*, 22nd May 2007.

[HBMB05] Graeme S. Halford, Rosemary Baker, Julie E. McCredden, and John D. Bain. How many variables can humans process? *Psychological Science*, 16(1):70–76, 2005.

[HD05] Sarjoughian Hessam and Huang Dongping. A multi-formalism modeling composability framework: agent and discrete-event models. In *Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*, pages 249–256, 2005.

[Hei02] Harald Heinecke. Open system architectures - the key to handle the complexity of upcoming automotive e/e-architectures. In *VDA Congress "Safety through Electronics"*, Proceedings of the Technical Congress 2002, pages 267 – 276, Stuttgart, Germany, 2002.

[HHK01] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: A time-triggered language for embedded programming. In *Proceedings of the First International Workshop on Embedded Software (EMSOFT)*, pages 166–184. Springer-Verlag, 2001.

[HK02] Thomas A. Henzinger and Christoph M. Kirsch. The embedded machine: Predictable, portable real-time code. In *Proceedings of the International Conference on Programming Language Design and Implementation (PLDI)*, pages 315–326. ACM Press, 2002.

[HKS05] Thomas A. Henzinger, Christoph M. Kirsch, and Matic Slobodan. Composable code generation for distributed Giotto. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 21–30, Chicago, Illinois, USA, 2005.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language Lustre. *Software Engineering, IEEE Transactions on*, 18(9):785–793, 1992.

[HLTW03] Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, 2003.

[HMKD82] W. Harrison, K. Magel, R. Kluczny, and A. DeKock. Applying software complexity metrics to program maintenance. *Computer*, 15(9):65–79, 1982.

[HOP06] Bernhard Huber, Roman Obermaisser, and Philipp Peti. MDA-based development in the DECOS integrated architecture - modeling the hardware platform. In *9th IEEE International Symposium on Object and component-oriented Real-time distributed Computing (ISORC)*, 2006.

[HS07] Thomas A. Henzinger and Joseph Sifakis. The discipline of embedded systems design. *IEEE Computer*, 40(10):32–40, 2007.

[HSF+04] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundth, Jean Leflour, Jean-Luc Maté, Kenji Nishikawa, and Thomas Scharnhorst. Automotive open system architecture - an industry-wide initiative to manage the complexity of emerging automotive E/E-architecture, 2004.

[HSLGM07] W. Herzner, R. Schlick, A. Le Guennec, and B. Martin. Model-based simulation of distributed real-time applications. In *5th IEEE International Conference on Industrial Informatics*, volume 2, pages 989–994, 2007.

[Hua05] Jinfeng Huang. *Predictability in Real-Time Software Design.* PhD thesis, 2005.

[HV06] F. Herrera and E. Villar. A framework for embedded system specification under different models of computation in SystemC. In *Model Based Verification of SystemC Designs Design Automation Conference*, pages 911–914, 2006.

[HVF+05] Jinfeng Huang, Jeroen Voeten, Oana Florescu, Piet van der Putten, and Henk Corporaal. Predictability in real-time system development. In *Advances in Design and Specification Languages for SoCs*, pages 123–139. Springer US, 2005.

[HVG+07] Fernando Herrera, Eugenio Villar, Christoph Grimm, M. Damm, and J. Haase. Heterogeneous specification with HetSC and SystemC-AMS: Widening the support of MoCs in SystemC. In Eugenio Villar, editor, *Forum on Design Languages (FDL)*. Springer, 2007.

[iec] IEC 60601: Medical electrical equipment.

[iec98] IEC 61508-4: Definitions and abbreviations, 1998-12 1998.

[IEC00] IEC 61508-2: Requirements for electrical / electronic / programmable electronic safety-related systems, 2000 2000.

[iee85] *IEEE 754: Standard for Binary Floating-Point Arithmetic*. IEEE, 1985.

[IEE94] IEEE standard VHDL language reference manual, 1994.

[iee04] IEEE 1588 / IEC 61588: : IEEE precision clock synchronization protocol for networked measurement and control systems, 2004.

[iee05] IEEE 1666: Standard SystemC language reference manual, 2005.

[IF00] Damir Isovic and Gerhard Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *IEEE Real-Time Systems Symposium*, 2000.

[iso04] ISO 8601:2004 - data elements and interchange formats – information interchange – representation of dates and times. Technical report, ISO, 12th January 2004.

[J.00] Barbour J. *The End of Time - The next revolution of physics*. Oxford University Press, New York City, New York (USA), 2000.

[JSPP04] Mirko Jakovljevic, Martin Schlager, Markus Plankensteiner, and Stefan Poledna. A path to mature safety - relevant automotive electronic solutions. *Automotive Electronics*, pages 2–4, 2004.

[JT06] Kim Jungin and B. Thuraisingham. Dependable and secure TMO scheme. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, page 8 pp., 2006.

[JTM07] Daniel Jackson, Martyn Thomas, and Lynette I. Millett. *Software for Dependable Systems: Sufficient Evidence?* National Academies Press, 2007.

[JVG03] Huang Jinfeng, J. Voeten, and M. Geilen. Real-time property preservation in approximations of timed systems. In *First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 163–171, 2003.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (TTE) design. In *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, Seattle, Washington, 2005.

[KAVS08] Ali Koudri, Denis Aulagnier, Didier Vojtisek, and Philippe Soulard. Using MARTE in a co-design methodology. In *Design, Automation and Test in Europe (DATE)*, 2008.

[KB03] H. Kopetz and G. Bauer. The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software*, 91:112–126, 2003.

[KBH+07] M. Krause, O. Bringmann, A. Hergenhan, G. Tabanoglu, and W. Rosentiel. Timing simulation of interconnected AUTOSAR software-components. In *Design, Automation, Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2007.

[KESHO07] H. Kopetz, C. El-Salloum, B. Huber, and R. A. Obermaisser R. Obermaisser. Periodic finite-state machines. In C. El-Salloum, editor, *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 10–20, 2007.

[KJ05] K. H. (Kane) Kim and Stephen F. Jenks. Facilitating distributed time-triggered simulation of embedded systems and environments. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 228b–228b, 2005.

[KOESH07] H. Kopetz, R. Obermaisser, C. El Salloum, and B. Huber. Automotive software development for a multi-core system-on-a-chip. In *Fourth International Workshop on Software Engineering for Automotive Systems (ICSE Workshops SEAS)*, pages 2–2, 2007.

[Kop97] Hermann Kopetz. *Design Principles for Distributed Embedded Applications*. Kluwer Academic Publisher, 1997.

[Kop98] Hermann Kopetz. The time-triggered model of computation. In *19th IEEE Systems Symposium (RTSS)*, 1998.

[Kop00a] Hermann Kopetz. Composability in the time-triggered architecture. In *SAE International Congress and Exhibition (2000-01-1382)*, Detroit, MI, USA, 2000.

[Kop00b] Hermann Kopetz. Software engineering for real-time: A roadmap. In *International Conference on Software Engineering*, pages 201–211, Limerick, Ireland, 2000.

[Kop03] H. Kopetz. The future of autonomous decentralized systems. page 329, 2003.

[Kop04] Hermann Kopetz. Composable embedded systems. In *Second IEEE International Conference on Computational Cybernetics (ICCC)*, pages 3–3, 2004.

[Kop06] Hermann Kopetz. Architectural issues in dependable embedded systems. In *13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB)*, Nuremberg, Germany, 2006.

[Kop07] Hermann Kopetz. The complexity challenge in embedded system design. Technical Report 55/2007, TU Wien, 17 September 2007.

[Kop08a] H. Kopetz. The complexity challenge in embedded system design. In *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12, 2008.

[Kop08b] Hermann Kopetz. The rationale for time-triggered ethernet. In *Real-Time Systems Symposium (RTSS)*, Barcelona, Spain, 2008.

[KOPS04] Hermann Kopetz, Roman Obermaisser, Philipp Peti, and Neeraj Suri. From a federated to an integrated architecture for dependable real-time embedded systems. Technical report, TU Wien, September 2004.

[Kor06] Andreas Korff. AUTOSAR system description with adapted UML 2.0 syntax. *ECE - Embedded Control Europe*, pages 33–35, 2006.

[KS03] Hermann Kopetz and Neeraj Suri. Compositional design of rt systems: A conceptual basis for specification of linking interfaces. In *In Proc. 6th IEEE International Symposium on Object-oriented Real-Time Distributed Computing (ISORC)*, Hokkaido, Japan, 2003.

[KSHP02] Christoph M. Kirsch, Marco A.A. Sanvido, Thomas A. Henzinger, and Wolfgang Pree. A Giotto-based helicopter control system. *Proceedings of the Second International Conference on Embedded Software*, 2491:46–60, 2002.

[KT98] Hermann Kopetz and Thomas Thurner. TTP – a new approach to solving the interoperability problem of independently developed ECUs, 1998.

[Kur05] I. Kurtev. *Adaptability of Model Transformations*. Phd thesis, 2005.

[Kyu08] Kita Kyushu. Clock constraint specification language (presentation). Technical report, 2008.

[Lar04] Magnus Larsson. *Predicting Quality Attributes in Component-based Software Systems*. Phd, 2004.

[LBH06] X. Li, M. Boldt, and R. Hanxleden. Mapping Esterel onto a multi-threaded embedded processor. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, 2006.

[Lee99] Edward A. Lee. Embedded software - an agenda for research. Technical report, University of California, Berkeley, 1999.

[Lee05] E. A. Lee. Absolutely positively on time: what would it take? [embedded computing systems]. *Computer*, 38(7):85–87, 2005.

[Lee06] Edward A. Lee. The future of embedded software. In *ARTEMIS 2006 Annual Conference*, Graz, Austria, 2006.

[LMZO03] Penny Lewis, R. C. Miall, P. Zlomanczuk, and G. Overkamp. Interval timing does not rely upon the circadian pacemaker. *Neuroscience Letters*, 348(3):131–134, 2003.

[LT07] Yanguo Liu and Issa Traore. Complexity measures for secure service-oriented software architectures. In Issa Traore, editor, *International Workshop on Predictor Models in Software Engineering (PROMISE)*, pages 11–11, 2007.

[LW05] Penny Lewis and V Walsh. Time perception: Components of the brain's clock. *Current Biology*, 15(10):389–391, 2005.

[LWS07] Alexandra C. Livesey, Matthew B. Wall, and Andrew T. Smith. Time perception: Manipulation of task difficulty dissociates clock functions from other cognitive demands. *Neuropsychologia*, 45(2):321–331, 2007.

[LZC+08] A. D. Ludlow, T. Zelevinsky, G. K. Campbell, S. Blatt, M. M. Boyd, M. H. G. de Miranda, M. J. Martin, J. W. Thomsen, S. M. Foreman, Jun Ye, T. M. Fortier, J. E. Stalnaker, S. A. Diddams, Y. Le Coq, Z. W. Barber, N. Poli, N. D. Lemke, K. M. Beck, and C. W. Oates. Sr lattice clock at 1 x 10-16 fractional uncertainty by remote optical evaluation with a Ca clock. *Science Magazine*, 318(5871):1805–1808, 28th March 2008.

[Mar03] Peter Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[MAR+08] Monica Malvezzi, Benedetto Allota, Mirko Rinchi, Michele Bruzzo, and Paola De Bernardi. Odometric estimation for automatic train protection and control systems. In *8th World Congress on Railway Research (WCRR)*, Korea, 2008.

[mat07] mathworld.wolfram.com, 2007.

[MB04] Michael D. Mauk and Dean V. Buonomano. The neural basis of temporal processing. *Annual review of neuroscience*, 27:307–340, 2004.

[MBSP02] Reinhard Maier, G. Bauer, Georg S., and Stefan Poledna. Time-triggered architecture - a consistent computing platform. In *IIE micro*, volume 4, pages 36–45, 2002.

[MCIJ+02] Gaudel M.-C., V. Issarny, C. Jones, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and F. Taiani. Final version of the DSoS conceptual model. Technical Report 54/2002, DSoS, 2002.

[MDHH05] E. Markert, M. Dienel, G. Herrmann, and U. Heinkel. System model of an inertial navigation system using SystemC-AMS. 2005.

[Mer05] David N. Mermin. *It's about time: understanding Einstein's relativity*. Princeton University Press, 2005.

[MHF04] G. Menkhaus, M. Holzmann, and S. Fischmeister. Time-triggered communication for distributed control applications in a timed computational model. In *The 23rd Digital Avionics Systems Conference (DASC)*, volume 2, pages 9.B.2–91–12, 2004.

[MHJGK+00] Kim Moon Hae, Kim Jung-Guk, K. H. Kim, Lee Myeong-Soo, and Park Shin-Yeol. Time-triggered message-triggered object modeling of a distributed real-time control application for its real-time simulation. In *24th International Computer Software and Applications Conference*, pages 549–556, 2000.

[Mil06] David L. Mills. Network time protocol version 4 - reference and implementation guide. Technical Report 06-6-1, University of Delaware (NTP Working Group), June 2006.

[Moa97] R. Moats. RFC2141 - URN syntax. Technical report, The Internet Engineering Task Force (IETF), May 1997.

[Mon07] David Monniaux. The pitfalls of verifying floating-point computations. Technical report, April 20 2007.

[MPP07] Marcello Mura, Amrit Panda, and Mauro Prevostini. Executable models and verification from MARTE and SysML: a comparative study of code generation capabilities. In *Design, Automation and Test in Europe (DATE) - Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile*, pages 29–34, Munich, Germany, 2007.

[MTAC01] M. Malvezzi, P. Toni, B. Allotta, and V. Colla. Train speed and position evaluation using wheel velocity measurements. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, volume 1, pages 220–224, 2001.

[Nel90] Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, 1990.

[NZTWF04] K. D. Nguyen, Sun Zhenxin, P. S. Thiagarajan, and Wong Weng-Fai. Model-driven SoC design via executable UML to SystemC. In *25th IEEE International Real-Time Systems Symposium*, pages 459–468, 2004.

[OESHK07] R. Obermaisser, C. El-Salloum, B. Huber, and H. Kopetz. Modeling and verification of distributed real-time systems using periodic finite state machines. *Journal of Computer Systems Science & Engineering*, 2007.

[OESHK08] Roman Obermaisser, Christian El Salloum, Bernhard Huber, and Hermann Kopetz. The time-triggered system-on-a-chip architecture. In *IEEE International Symposium on Industrial Electronics (ISIE)*, Cambridge, 2008.

[Oga01] Katsuhiko Ogata. *Modern Control Engineering*. Prenctice Hall, 2001.

[OHK+05] Roman Obermaisser, E. Henrich, K.H. Kim, Hermann Kopetz, and M.H. Kim. Integration of two complementary time-triggered technologies: TMO and TTP. In *IFIP International Federation for Information Processing*, Computer Science, pages 211–222. Springer Boston, 2005.

[OK09] Roman Obermaisser and Hermann Kopetz. *GENESYS An ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Sudwestdeutscher Verlag fur Hochschulschriften, 2009.

[OMG01] OMG. Smart transducers interface, 2001.

[omg08] A UML profile for MARTE: Modeling and analysis of real-time embedded systems, 2008.

[Oxf07] Dictionaries Oxford. Oxford english dictionary, 2007.

[PAAP10] Jon Perez, Mikel Azkarate-Askasua, and Antonio Perez. Codesign and simulated fault injection of safety-critical embedded systems using SystemC. In *Eighth European Dependable Computing Conference (EDCC)*, Valencia, Spain, 2010.

[Pan01] P. R. Panda. SystemC - a modeling platform supporting multiple design abstractions. In *The 14th International Symposium on System Synthesis*, pages 75–80, 2001.

[Par01] Daisy Erin Parker. Floating-point from three perspectives. Technical report, April 23, 2001 2001.

[Pas99] N. Paskin. Toward unique identifiers. *Proceedings of the IEEE*, 87(7):1208–1227, 1999.

[PB00] Peter Puschner and Alan Burns. A review of worst-case execution-time analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, 2000.

[PBdST04] Dumitru Potop-Butucaru, Robert de Simone, and Jean-Pierre Talpin. The synchronous hypothesis and synchronous languages. page 21, 2004.

[PBKS07] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *Future of Software Engineering (FOSE)*, pages 55–71, 2007.

[PBWB00] Stefan Poledna, Alan Burns, Andy Wellings, and Peter Barrett. Replica determinism and flexible scheduling in hard real-time dependable systems. *Transactions on Computers*, 49(2):100–111, 2000.

[PC06] Juan M. Perez Cerrolaza. Safety-critical transportation embedded-systems (state of the art). Technical report, Ikerlan, 16th June 2006.

[PC07a] Juan M. Perez Cerrolaza. Composability concept for embedded systems. Technical report, Ikerlan - TU Wien, 2007.

[PC07b] Juan M. Perez Cerrolaza. Determinism concept for embedded systems. Technical report, Ikerlan - TU Wien, 2007.

[PdAEOSS+08] Igor Perez-de Arenaza, Ion Etxeberria-Otadui, Jon San-Sebastian, Unai Viscarret, Amaia Lopez-de Heredia, and J. M. Azurmendi. Plataforma experimental de evaluación IGCTs para aplicaciones de multi-megavatio. In *Seminario Anual de Automática, Electrónica Industrial e Instrumentación (SAAEI)*, Cartagena, Spain, 2008.

[PH07] Isabelle Puaut and Damien Hardy. Predictable paging in real-time systems: A compiler approach. In *19th Euromicro Conference on Real-Time Systems (ECRTS '07)*, pages 169–178, 2007.

[PKjE06] Becky Plummer, Mukul Kha janchi, and Stephen A. Edwards. An esterel virtual machine for embedded systems. In *Synchronous Languages, Applications, and Programming (SLAP)*, 2006.

[PLC08] PLCopen. IEC 61131-3 tutorial. Technical report, 2008.

[PNOES10] Jon Perez, Carlos Fernando Nicolas, Roman Obermasser, and Christian El Salloum. Modeling time-triggered architecture based safety-critical embedded systems using systemc. In *Forum on specification & Design Languages (FDL)*, 2010.

[POT+05] Philipp Peti, Roman Obermaisser, F. Tagliabo, A. Marino, and S Cerchio. An integrated architecture for future car generations. In *Object-oriented Real-time distributed Computing (ISORC)*, pages 2–13, Seattle, Washington, USA, 2005.

[PPO10] Jon Perez, Antonio Perez, and Roman Obermaisser. Executable time-triggered model (E-TTM) for real-time control systems. In *13th Symposium on Object-oriented Real-time distributed Computing (ISORC)*, Carmona, Spain, 2010.

[PSPG00] V. Padmanabhan, A. Sattar, A. K. Pujari, and C. Goswamy. Temporal reasoning: a three way analysis. In *Seventh International Workshop on Temporal Representation and Reasoning (TIME)*, pages 183–189, 2000.

[PT05] Wolfgang Pree and Josef Templ. Modeling with TDL. Technical report, University of Salzburg, August 2005.

[PW03a] Mikel D. Petty and Eric W. Weisel. A composability lexicon. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, pages 181–187, 2003.

[PW03b] Mikel D. Petty and Eric W. Weisel. A formal basis for a theory of composability. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, pages 181–187, 2003.

[Ram99] T. H. Rammsayer. Neuropharmacological evidence for different timing mechanisms in humans. *The Quarterly Journal of Experimental Psychology B*, 52(3):273–286, 1999.

[RE06] Bernhard Rumpler and Wilfried Elmenreich. Considerations on the complexity of embedded real-time system design tasks. In *IEEE International Conference on Computational Cybernetics (ICCC)*, Tallinn, Estonia, 2006.

[Ric06] Kai Richter. The AUTOSAR timing model - status and challenges. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 9–10, 2006.

[Rie00] Heijo Rieckmann. Managen und führen am rande des 3. jahrtausends. In *Europäischer Verlag der Wissenschaften*, Frankfurt, 2000.

[RL94] Ha Rhan and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 162–171, 1994.

[RS03] Christine Richange and Pascal Sainrat. Towards designing WCET-predictable processors. In *5th Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2003.

[RSD04] Partha S. Roop, Zoran Salcic, and M. W. Sajeewa Dayaratne. Towards direct execution of esterel programs on reactive processors. In *Proceedings of the 4th ACM international conference on Embedded software*, 2004.

[Rum06] Bernhard Rumpler. Complexity management for composable real-time systems. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, page 9 pp., 2006.

[SAS+07] Erwin Schoitsch, Egbert Althammer, Gerald Sonneck, Henrik Eriksson, and Jonny Vinter. Support for modular certification of safety-critical embedded systems in DECOS - the generic safety case*. In *ERCIM / DECOS Workshop on "Dependable Embedded Systems"*, 2007.

[Sch03] Eric Schwarz. Revisions to the IEEE 754 standard for floating-point arithmetic. In *16th IEEE Symposium on Computer Arithmetic*, pages 112–112, 2003.

[SG07] D. Schreiner and K. M. Goschka. A component model for the AUTOSAR virtual function bus. In *31st Annual International Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 635–641, 2007.

[SGAK06] Klaus Steinhammer, Petr Grillinger, Astrit Ademaj, and Hermann Kopetz. A time-triggered ethernet (TTE) switch. In

Design, Automation and Test in Europe (DATE), volume 1, pages 1–6, 2006.

[SKMVM04] Neeraj Suri, Hermann Kopetz, Mirek Malek, and Aad Van Moorsel. Highly available composable services. reality or wishful thinking? In *ISAS*, 2004.

[SLMR05] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, 38(11):23–31, 2005.

[SM03] B. Selic and L. Motus. Using models in real-time software design. *IEEE Control Systems Magazine*, 23(3):31–42, 2003.

[Sno95] Richard T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 7:513–532, 1995.

[Sou06] Thomas Soulier. Software business models for AUTOSAR automotive world standard (v1.00). In *3rdEuropean congress ERTS - Embedded Real Time Software*, page 10, Tolouse (France), 2006.

[Spa04] D. Spahni. Managing access to distributed resources. In *37th Annual Hawaii International Conference on System Sciences*, page 8 pp., 2004.

[SR90] John A. Stankovic and Krithi Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, 1990. review again.

[SS75] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SS01] Dr. Janos Sztipanovits and Shankar Sastry. Embedded software: Opportunities and challenges. Technical report, DoD & DARPA, 23-26 June 2002 2001.

[SS03] André Seznec and Nicolas Sendrier. HAVEGE: A user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation (TOMACS*, 13(4):334–346, 2003.

[SS04] Christian Salzmann and Thomas Stauner. Automotive software engineering. In *Languages for System Specification*, pages 333–347. Springer US, 2004.

[SSC+04] N. Scaife, Christos Sofronis, Paul Caspi, Stavros Tripakis, and F. Maraninchi. Defining and translating a "safe" subset of Simulink/Stateflow into Lustre, July 2004.

[ST07] Claudia Szabo and Yong Meng Teo. On syntactic composability and model reuse. In *First Asia International Conference on Modelling & Simulation (AMS)*, pages 230–237, 2007.

[ST08] Dario D. Salvucci and Niels A. Taagten. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review*, 115(1):101–130, 2008.

[Sta05] J. E. R Staddon. Interval timing: memory, not a clock. *Trends in cognitive sciences*, 9(7):312–314, 2005.

[std01] IEC 61513: Nuclear power plants - instrumentation and control for systems important to safety - general requirements for systems, 2001.

[std04] IEC 61511: Functional safety - safety instrumented systems for the process industry sector, 2004.

[std05] IEC 62061: Safety of machinery - functional safety of safety-related electrical, electronic and programmable electronic control systems, 2005.

[Sur06] Neeraj Suri. A line in the sand: Emerging embedded system challenge, 2006.

[sys06] SysML specification v1.0 (draft), 2006.

[Sze05] Tivadar Szemethy. Case study: Model transformations for time-triggered languages. In *International Workshop on Graph and Model Transformation (GraMot)*, Tallinn, Estonia, 2005.

[Szt00] Dr. Janos Sztipanovits. Embedded software: Opportunities and challenges. In *DARPATECH*, Dallas, Texas, USA, 2000.

[TC03] Stavros Tripakis and P. Caspi. Issues in realizing an end-to-end embedded system tool-chain: Experiences from the european ist projects “NEXT TTA” and “RISE”, 2003.

[Tem05] Josef Templ. The timing definition language (TDL). Technical report, Universität Salzburg, 16th January 2005.

[Tem07] Josef Templ. TDL specification and language report. Report, preeTE - Wolfgang Pree GmbH, 2007.

[THH02] H. Tada, O. Honda, and M. Higuchi. A file naming scheme using hierarchical-keywords. In *26th Annual International Computer Software and Applications Conference (COMPSAC)*, pages 799–804, 2002.

[Tru07] Salvador Trujillo. *Feature Oriented Model Driven Product Lines*. Phd thesis, 2007.

[TW04] Lothar Thiele and Reinhard Wilhelm. Design for time-predictability. In Lothar Thiele Wilhelm and Reinhard, editors, *Perspectives Workshop: Design of Systems with Predictable Behaviour*, Dagstuhl Seminar Proceedings, 2004.

[VGE03a] A. Vachoux, C. Grimm, and K. Einwich. Analog and mixed signal modelling with SystemC-AMS. In *International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 914–917, 2003.

[VGE03b] A. Vachoux, C. Grimm, and K. Einwich. SystemC-AMS requirements, design objectives and rationale. In *Design, Automation and Test in Europe*, pages 388–393, 2003.

[VGE04] Alain Vachoux, Christoph Grimm, and Karsten Einwich. Towards analog and mixed-signal soc design with SystemC-AMS. In *Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Application*, 2004.

[VH08] Eugenio Villar and Fernando Herrera. HetSC system specification in SystemC. In *SATURN - Embedded Systems Design Workshop*, Athens, 2008.

[vRT08] Hedderik van Rijn and Niels A. Taagten. Timing of multiple overlapping intervals: How many clocks do we have? *Acta Psychologica*, 129(3):365–375, 2008.

[Vya07] Valeriy Vyatkin. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. ISA - Instrumentation, Systems, and Automation Society, 1 edition, 2007.

[W3C07] W3C. Web services description language (WSDL) 2.0 - part 1: Core language. Technical report, 2007.

[wa] www.systemc ams.org.

[Web89] Webster. *Encyclopaedic Dictionary*. 1989.

[Win09] Peter Winter. *Compendium on ERTMS - European Railway Traffic Management System*. DW Media Group GmbH, 2009.

[WKPR05] Ingomar Wenzel, Raimund Kirner, Peter Puschner, and Bernhard Rieder. Principles of timing anomalies in superscalar processors. In *Fifth International Conference on Quality Software (QSIC)*, pages 295–303, 2005.

[WR94] T. Wahl and K. Rothermel. Representing time in multimedia systems. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 538–543, 1994.

[WRMS03] Matthias Werner, Jan Richlingz, Nikola Milanovicz, and Vladimir Stantchevz. Composability concept for dependable embedded systems. In *22nd Symposium on Reliable Distributed Systems (SRDS)*, Florence, Italy, 2003.

[wt] www.esterel technologies.com.

[WT06] Reinhard Wilhelm and Lothar Thiele. Timing predictability - a must for avionics systems. In *National Workshop on Aviation Software Systems: Design for Certifiably Dependable Systems*, Alexandria, VA, 2006.

[wwwa] www.autosar.org.

[wwwb] www.bipm.fr/en/scientific/tai/tai.html.

[wwwc] www.decos.at.

[wwwd] www.iec.ch.

[wwwe] www.mathworks.com.

[wwwf] www.sysml.org.

[wwwg] www.systemc.org.

[wwwh] www.timmo.org.

[wwwi] www.tttech.com.

[wwwj] www.uml.org.

[www07] www.w3.org/TR/wsdl20. Web services description language (WSDL) 2.0 - part 1: Core language, 2007.

[XP90] Jia Xu and David Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, 1990.

[XP93] J. Xu and D. L. Parnas. On satisfying timing constraints in hard-real-time systems. *IEEE Transactions on Software Engineering*, 19(1):70–84, 1993.

[XSH+06] Piao Xuefeng, Han Sangchul, Kim Heeheon, Park Minkyu, Cho Yookun, and Cho Seongje. Predictability of earliest deadline zero laxity algorithm for multiprocessor real-time systems. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, page 6, 2006.

[Xu03] Jia Xu. Making software timing properties easier to inspect and verify. *IEEE Software*, 20(4):34–41, 2003.

[Yin08] Wang Yingxu. The cognitive processes of perceptions on spatiality, time, and motion. In *7th IEEE International Conference on Cognitive Informatics (ICCI)*, pages 239–248, 2008.

[YJ03] Wang Yingxu and Shao Jingqiu. Measurement of the cognitive functional complexity of software. pages 67–74, 2003.

[YJL07] Zhao Yang, Liu Jie, and E. A. Lee. A programming model for time-synchronized distributed real-time systems. In *Real Time and Embedded Technology and Applications Symposium (RTAS)*, pages 259–268, 2007.

[YXGB+06] Wang Ying, Zhou Xue-Gong, Zhou Bo, Liang Liang, and Peng Cheng-Lian. A MDA based SoC modeling approach using UML and SystemC. In *The Sixth IEEE International Conference on Computer and Information Technology (CIT)*, pages 245–245, 2006.

[ZLL04] Jianfan Zou, David Levy, and Anna Liu. Evaluating overhead and predictability of a real-time CORBA system. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.

Publications

- Jon Perez, Carlos Fernando Nicolas, Roman Obermaisser, Christian El Salloum **Modeling Time-Triggered Architecture Based Safety-Critical Embedded Systems Using SystemC**. *Forum on specification & Design Languages (FDL), 2010, Southampton, UK.*

- Jon Perez, Antonio Perez, Roman Obermaisser **Executable Time-Triggered Model (E-TTM) for Real-Time Control Systems**. *The 13th IEEE International Symposium on Object/component/service-oriented Real-time distributed computing (ISORC), 2010, Spain.*

- Jon Perez, Mikel Azkarare-Askasua, Antonio Perez. **Codesign and simulated fault injection of safety-critical embedded systems using SystemC**. *8th European Dependable Computing Conference (EDCC), 2010, Spain.*

- Xabier Iturbe, Mikel Azkarate, Imanol Martinez, Jon Perez, and Armando Astarloa. **A novel SEU, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs**. *19th International Conference on Field Programmable Logic and applications (FPL), 2009, Czech Republic. pages 569-573, ISSN: 1946-1488*

- J.M. Perez Cerrolaza, Oskar Berreteaga, Alberto Ruiz de Olano **A Modeling Framework for Efficient Safety Critical Time-Triggered Architecture Design**. *Society of Automotive Engineering (SAE), 2007, Detroit (USA). numb 2129, pages 1-12, ISSN: 1533-6204*

- Josu Bilbao, Mikel Zorrilla, Gorka Epelde, Juan Martin Perez, Igor Armendariz. **Industrial Ethernet architectures to provide QoS and add a convergent prospect for in-home High Definition Multimedia service distribution**. *46th Forum for European ICT Professionals (FITCE), 2007, Warsaw.*

- J.M. Perez Cerrolaza, Oskar Berreteaga, Alberto Ruiz de Olano, Antonio Perez, Amaia Urkidi **A methodology for developing embedded hardware and software for critical safety-systems**, *Mikroelektronik ME, 2006, Vienna.*

- Antonio Perez, Amaia Urkidi, Oskar Berreteaga, Alberto Ruiz de Olano, Perez Juan Martin. **Metodologia para el desarrollo de hardware / software embebido en sistemas criticos de seguridad**. *Mundo Electronico, 2006, p. 42-48, ISSN: 0300-3787*

Curriculum Vitae

	Details
Name:	Juan Martin (Jon), Perez Cerrolaza `jmperez@ikerlan.es`
Sex:	Male
Nationality:	Spanish
Birth:	1975-03-31

	Education
2006 - present	Vienna University of Technology (TU Wien, Austria)
	Institute of Computer Engineering, Real-Time Systems Group
	PhD: "Executable Time-Triggered Model (E-TTM) for the development of safety-critical embedded systems"
1999 - 2000	University of Glasgow (UK)
	M.Sc. In Electronics & Electrical Engineering (with Distinction)
1996 - 1999	Mondragon Unibertsitatea (Spain)
	B.Eng. in Industrial Electronics and Robotics (6 years engineering)
1993 - 1996	Mondragon Unibertsitatea (Spain)
	Electronics and Electrical Engineering (4 years engineering)

	Professional Experience
2002 - present	Ikerlan Research Center (Mondragon, Spain)
	Real-time and safety-critical embedded systems developer (TÜV FS Eng ID-No. 2378/10), hardware and software
	Research focus on safety-critical embedded systems and certification (IEC-61508, EN-5012X, etc.)
2000 - 2002	Motorola SPS (East Kilbride, UK & Toulouse, France)
	DSP Applications Engineer
	Multi-core optimized real-time software developer
1998 - 1999	Mondragon Unibertsitatea (Mondragon, Spain)
	Postgraduate part-time & full-time job
	Develop machine-vision system for metal surface inspection