Accepted Manuscript

A Proposed Conceptual Architecture for Time-Sensitive Software-Systems

Frank J. Furrer[1]

[1]Technical University of Dresden, Germany. Faculty for Computer Science (frank.j.furrer@bluewin.ch)

Research Question:

«Time-Sensitive Software» (Lee et al., 2023)

Abstract

Many mission-critical systems today have stringent timing requirements. Especially for cyber-physical systems that directly interact with real-world entities, violating correct timing may cause accidents, damage, or endanger life, property, or the environment. To ensure the timely execution of time-sensitive software, a suitable system architecture is essential. This paper proposes a novel conceptual system architecture based on well-established technologies, including transition systems, process algebras, Petri Nets, and time-triggered communications. This architecture for time-sensitive software execution is described as a conceptual model backed by an extensive list of references and opens up several additional research topics. This paper focuses on the conceptual level and defers implementation issues to further research and subsequent publications.

This peer-reviewed article has been accepted for publication but not yet copyedited or typeset, and so may be subject to change during the production process. The article is considered published and may be cited using its DOI.

10.1017/cbp.2025.10002

Time in Computing

Time is a fascinating concept. Much has been thought and written about the physics of time (e.g., Muller, 2016), the philosophy of time (e.g., Power, 2021), the measurement of time (e.g., Struthers, 2024), and the history of time (e.g., Hawking, 2015). In computing, time has precise meanings (Furia, 2012 / Buttazzo, 2024), such as:

(1) The time elapsed between an event and the completion of the correct response (Latency);
(2) The maximum time guaranteed for a program to execute (Worst case execution time, WCET);
(3) The maximum time allowed for the execution of a process or a function (Before a time-out);
(4) The maximum time for a process to wait for an event, a response, or a message (Synchronization);
(5) The time interval between measurement values received from a sensor (Input sampling rate);
(6) The time interval between outputs to an actuator (Output sampling rate);
(7) The trigger times to start a process (Either absolute from UCT or relative to another event or process);
(8) Relative timing: Before, not before, after (For events, messages, actions, process start, etc.);
(9) … and other timing requirements or timing relationships.

Timing is a serious specification responsibility. In cyber-physical systems, strict adherence to correct timing requirements is a decisive safety property. Therefore, time-sensitive software is crucial for safety-critical cyber-physical systems!

State of the Art

The work on reference architectures for cyber-physical systems (e.g., Nakagawa et al., 2023) is not new. Several such architectures have been proposed and are well documented, e.g., generic architectures, such as: CPS 5 Components Architecture (Ahmadi et al., 2021), 8C architecture (Sony, 2020), NIST Framework for Cyber-Physical Systems (Griffor et al., 2017 / NIST, 2017). Or domain-specific architectures, such as: AUTOSAR (https://www.autosar.org/ Rajeev et al., 2012), IMA (Integrated Modular Avionics

Architecture, Gaska et al., 2015). Some architecture-centric standards, such as ISO 26262 (see, e.g., Debouk, 2019) and IEC 61499 (see, e.g., Thramboulidis, 2012; Yoong et al., 2013, 2016), are highly useful. However, these works treat timing as a *quality attribute* (= measurable or testable characteristics of a system, such as availability, reliability, usability, or scalability) and not as a *correctness property* of the system (= formal requirement that defines and assures the system's expected behavior), (Lee et al., 2023).

A different approach to handling time is the use of temporal logic. Many types of temporal logic systems exist (e.g., 16 of them are explained in Bellini et al., 2000). Temporal logic extends classical logic by defining temporal operators, allowing engineers to model and reason about the behavior of systems over time. Using temporal logic is a powerful methodology in software engineering, applicable to the specification, verification, and design of programs, algorithms, and databases (e.g., Bolc et al., 2019; Furia et al., 2012; Kröger et al., 2008). Temporal logic expresses timing well but cannot define and express the system architecture (Structure, relationships, attributes).

A different, generic, layered architecture has been proposed by Ungureanu et al. (2017). Their proposal utilizes different constructs, including the tagged signal model, the functional programming paradigm, and algorithmic skeletons. An additional framework is developed by Abdellatif et al. (2010) and Buckl et al. (2010), focusing on timing and safety.

The progress of this paper is a *conceptual* architecture with explicit, formalized, verifiable timing at all levels of the architecture and all steps of the lifecycle of the CPS:

I. Elevating timing from a *quality attribute* (= measurable or testable characteristics of a system, such as availability, reliability, usability, or scalability) to a *correctness property* of the system (= formal requirement that defines and assures the system's expected behavior);
II. Proposing a layered architecture that respects the proven, well-documented architecture principles, such as layering, partitioning, modularization, loose coupling, separation of concerns, etc. (Furrer, 2019 / Furrer, 2022);
III. Combines accepted constructs for timing definition, verification, and implementation (Process algebra, transition systems, Petri Nets, Time-Triggered Communications).

Introduction and Context

The context for time-sensitive software is shown in Figure 1. It consists of 6 elements:

1. The functional processes: These processes specify the functionality of the system. Note that the term is mainly used for business processes, but technical functionality is also represented as a (functional) process. The symbol τ represents the timing requirements of the process. Note that complete and correct error- and exception-handling is an indispensable and integral part of the processes (e.g., Öztemür, 2015);
2. The components (programs) implementing the functionality;
3. The execution platforms (processors, memory, communications, databases, etc.): Note that most of today's cyber-physical systems are distributed systems, i.e., they have more than one execution platform. Such systems are referred to as systems-of-systems (SoS). The different execution platforms communicate with each other – they are linked by one or several communication channels;
4. The interprocess-communication: The processes exchange information and flow control (such as synchronization, checkpoints);
5. Mechanism for the process orchestration. Start, stop, or interrupt processes, e.g., following an event, a message, a timing, or a schedule;
6. The connection to the real world: Sensors to read information, and actuators to control the physical world.

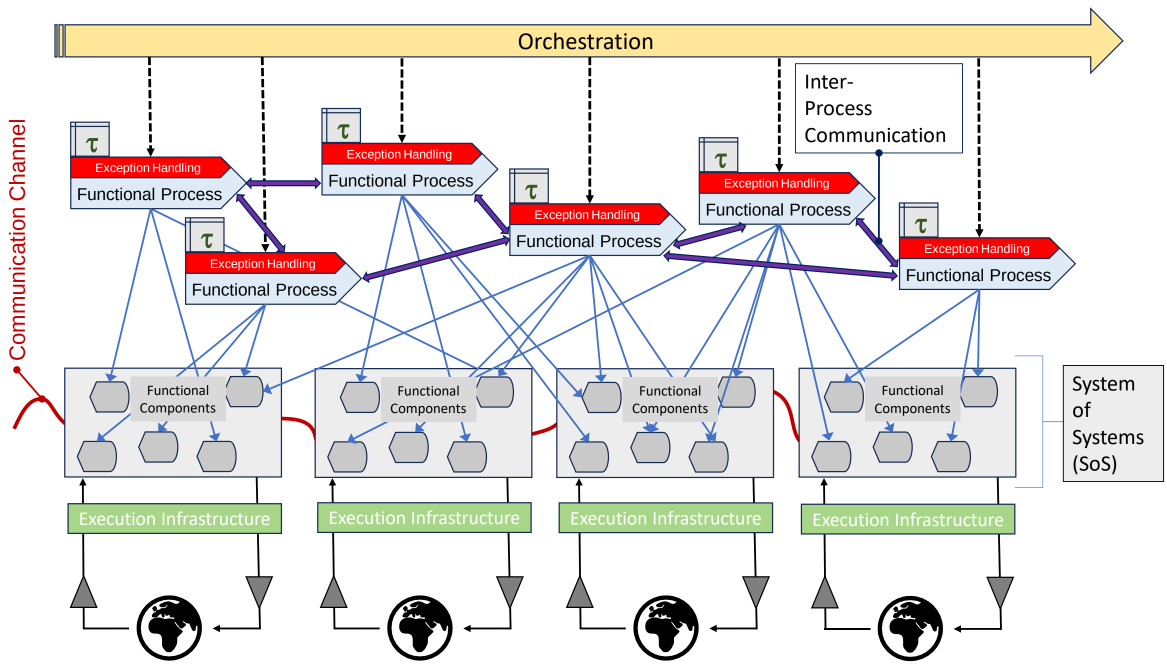
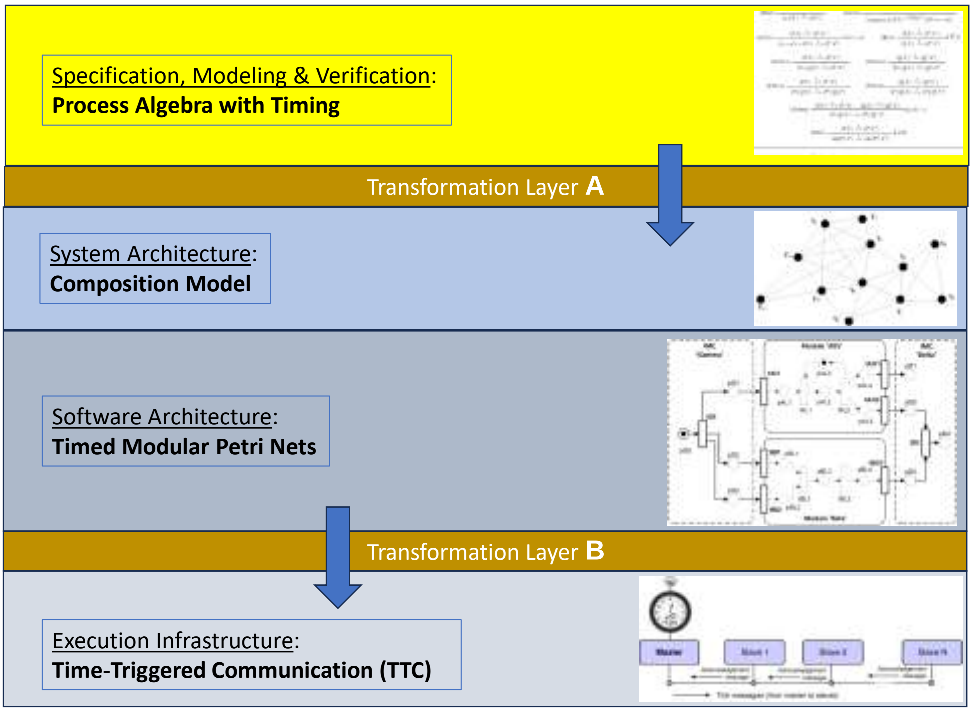


Figure 1: Context for Time-Sensitive Software

Layered Architecture Proposal

Context: All development and evolution mechanisms for time-sensitive software – from specification to operation – must have the proper constructs for correctly handling time. Unfortunately, most of today's methodologies and tools lack a consistent and verifiable handling of time – and are thus only of limited use for developing and verifying time-sensitive software.

Figure 2 is an attempt at a conceptual end-to-end architecture for time-sensitive software. Please note that this first sketch is a conceptual proposal and leaves open points for future research.



https://www.researchgate.net/figure/Simple-architecture-of-time-triggered-shared-clock-scheduler_fig1_308611848

Figure 2: Layered Architecture Proposal

Figure 2 proposes six architecture layers, each one with formal constructs to handle time explicitly:

I. *The specification, modeling, and verification layer* (Top layer): For this layer, a process algebra is used. Process algebra is a formal calculus for specifying, modeling, and verifying transition processes (DeNicola, 2011/Aldini et al., 2009/ Fokkink, 1999/Chao, 2015). Some process algebras include the formal constructs for timing (e.g., Baeten, 2001 /Baeten, 2002 / Wang, 2002 / Wolf, 2002);

II. *The system architecture layer*: Describes the parts (= components), their composition (= structure), and their relationships (= interactions). As a composition model, "Petri Nets for Modeling of Large Discrete Systems" (Davidrajuh, 2021) is utilized;

III. *The software architecture layer*: As the component model providing the functionality, "Petri Modules" and "Inter-modular connectors" (Davidrajuh, 2021) are selected. The Petri modules are enriched with timing constructs (Popova-Zeugmann, 2016 / Liu, 2022);

IV. *The execution infrastructure layer*: All software runs on the execution infrastructure layer. This layer encompasses all hardware, software systems, and communication elements. Again, an execution infrastructure that is time-aware, i.e., can provide execution timing guarantees, must be provided. The infrastructure of choice is the "Time-Triggered Communications" (Obermaisser et al., 2012 /Kopetz, 2022/Kopetz et al., 2003/Maier et al., 2002/ Rushby, 2005/Buttazzo, 2023);

V. In addition, two *transformation layers* are required. Transformation Layer A translates the verified specification model into the Petri Net specifications. Note that the system architecture (Petri Net structure) is designed before the transformation A. Transformation layer B maps the timed functionality of the Petri Nets to the TTA schedule, i.e., to the execution infrastructure.

Concurrency and Latency

The two most challenging topics in implementing time-critical CPS are *concurrency* (e.g., Gorrieri et al., 2015) and *latency* (e.g., Kopetz et al., 2022). In a modern CPS, many applications share common resources, such as CPUs, memory, external storage, and communications channels, i.e., parallel access to shared resources (Figure 3). This concurrency may result in one application or process influencing the timing of another application or process, sometimes adversarially, such that timing requirements may be violated, such as response times prolongated! If concurrency is not handled correctly, non-determinism can occur – delivering different results from a program run because of interference by concurrency (Gorrieri et al., 2015).

The second topic is latency (Figure 3): In a classical architecture implementation, there are many sources of latency: Operating system functions, scheduling, communications delays, shared memory access retardation, queuing, etc. Some of these delays may be unpredictable and can behave statistically. For dependable time-sensitive software, concurrency and latency must be identified, quantified, and adequately managed. The proposed architecture in Figure 2 is designed to strongly support this objective.

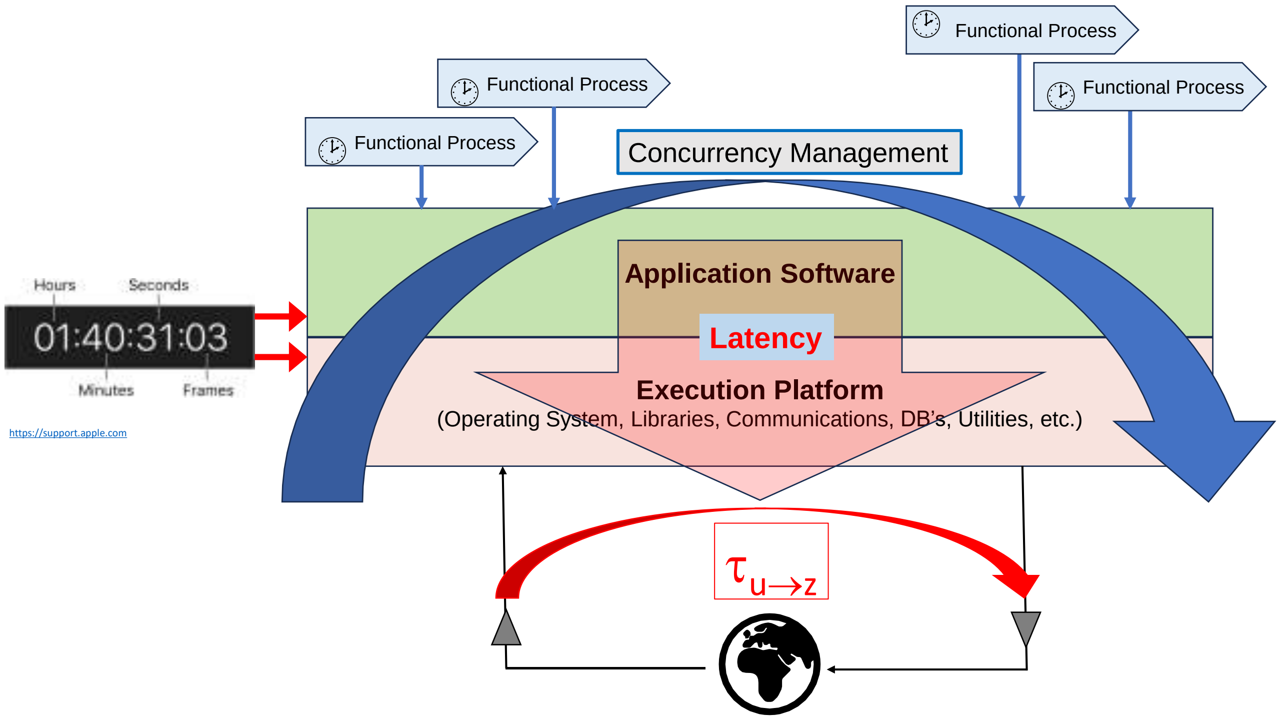


Figure 3: Concurrency and Latency in a Computing System

Process Algebra

Context: For the specification, verification, and modeling of the time-aware functional processes in the system (Top level layer of Figure 2), the methodology of Process Algebras with Time is chosen (e.g., Baeten, 2001/Baeten, 2002 / Wang, 2002). Process algebras are formalisms for specifying interactions (synchronization, flow control, semaphores, etc.) between concurrent processes. Modern process algebras evolved from the idea of formalizing communicating processes. The seminal contribution is the paper "A Calculus of Communicating Systems (CSS)" (Milner, 1980). In the following years, several new Process Algebras were developed (e.g., Baeten, 2005/Bergstra, 1984/Hoare, 1985). The early process algebras had no explicit and formal notion of timing. Timing was introduced later (e.g., Nicollin, 1991). Today, process algebras with fully formalized timing exist (e.g., Baeten, 2001 /Baeten, 2002 / Wang, 2002). A process algebra defines a set of operators for the interaction of concurrent processes. A process algebra with time has additional operators for formally handling time.

Many process algebras with rich literature are in use today (e.g., Aceto, 2003). So far, no favorite, widely accepted, and used process algebra exists. Process algebras are selected for the task at hand. For the widespread use of process algebras in industry, standardization by an industry body would be highly beneficial. A first attempt is the ISO standardization of a process algebra for communication protocols (Bolognesi et al., 1968 / ISO, 2001).

Transition Processes

Context: Process algebras require modeling the functionality of processes as transition systems (e.g., Demri et al., 2016; Gorrieri et al., 2015).

Transition systems have states. An action triggers the transition from one state to another. States and actions include explicit timing requirements in their specifications (Figure 4a, the symbol 🕑 represents the timing). The theory of state machines is well-known and provides sufficient formality (e.g., Börger et al., 2013).

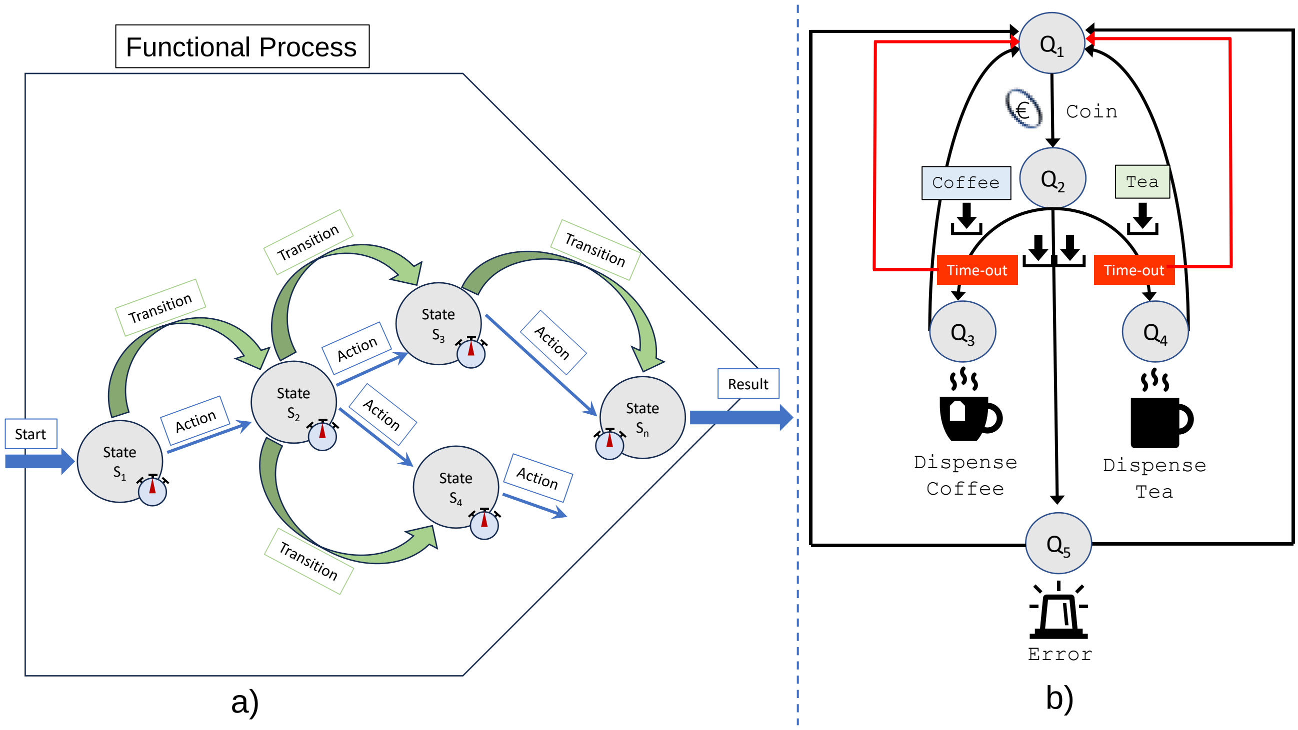


Figure 4: Transition Systems

Figure 4b shows the example of a vending machine that is often used as a (much simplified) transition system. It has five states: Q_1 (= «Waiting for coin»), Q_2 (= «Waiting selection»), Q_3 (= «Coffee»), Q_4 (= «Tea»), Q_5 (= «Error»). The transitions are represented by arrows, including time-out after coin insertion and pressing both buttons simultaneously.

Timed Petri Modules and Inter-Modular Connectors

Context: Several realizations of the Petri Net idea exist. The one best suited for this architecture has been developed by Reggie Davidrajuh (https://www.davidrajuh.net/reggie/). It is applicable to large discrete systems and allows arbitrary system structures.

The functionality and quality properties of the system are implemented using "Timed Petri Modules" (Popova-Zeugmann, 2016; Wang, 1998) and "Inter-Modular Connectors" (Davidrajuh, 2021, Figure 5).

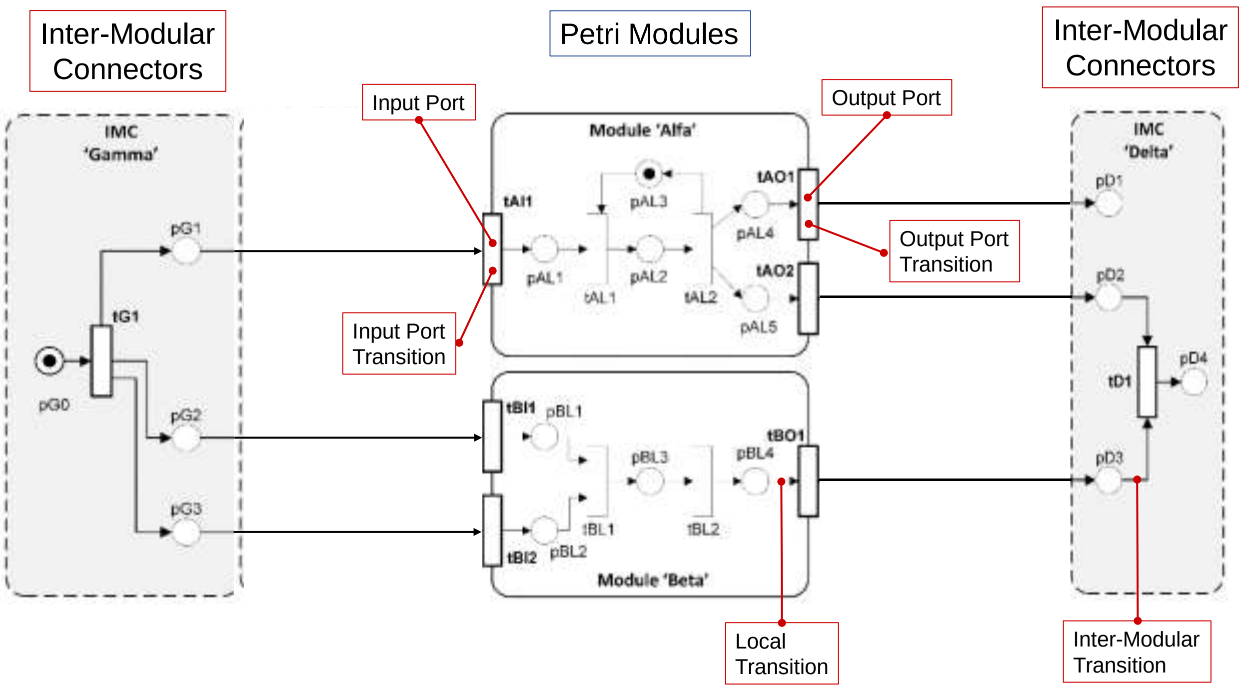


Figure 5: Timed Petri Modules and Inter-Modular Connectors

The Timed Petri Modules feature all the constructs and properties of Petri Modules with time (e.g., Girault, 2010). They implement the functionality and data. The interconnections of the Petri Modules specified by the process algebra are implemented by the Inter-Modular Connectors (IMC). These two building blocks give the architecture designer a high level of flexibility and allow any structure (not only hierarchical) to be defined.

The process algebra does not specify the system architecture. The distribution of functionality to the individual Petri Modules (Partitioning, cohesion, and coherence, etc.), the coupling of the Petri Modules by the Inter-Modular Connectors (Interfaces, loose coupling, etc.) must be designed by a specialized system/software architect. Fortunately, proven, well-documented architecture principles and patterns (Figure 6) are available to construct a dependable, maintainable, and evolvable architecture (e.g., Murer et al., 2014/Furrer, 2019/Furrer, 2022, Transformation Layer A below).

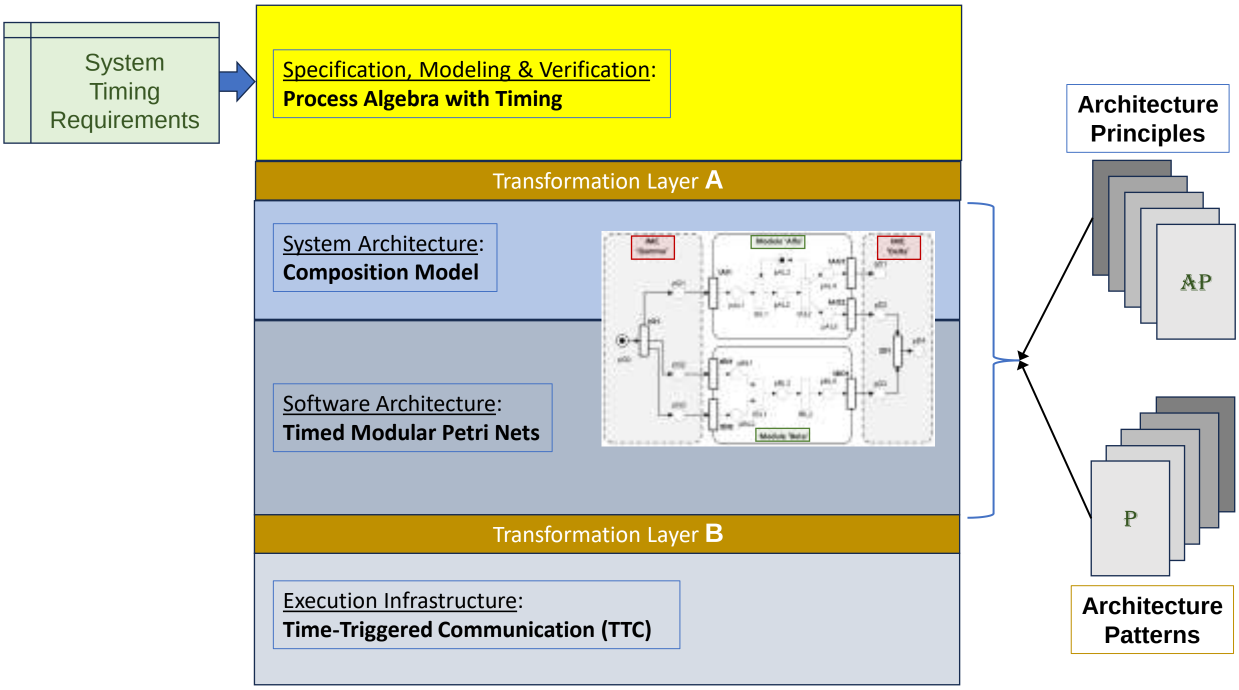


Figure 6: Software Architecture

Transformation Layer A

Context: While the four functional layers in Figure 6 use well-known, well-documented, and proven technologies (Such as transition processes, process algebras, timed Petri Nets, the IMC composition model, and time-triggered communications), the two transformation layers are new concepts. The transformation layer A maps a timed transition system onto a timed Modular Petri Net. Although some literature exists on this specific topic (e.g., Badouel et al., 2015 / Devillers et al., 2022 / Best et al., 2024 / Cortadella et al., 1995 / Goltz, 1990), this transformation layer becomes a research topic – especially concerning timing implementation.

The transformation layer A has two transformation paths (Figure 7):

Transformation Path 1 (Architecture, Figure 7):

The structural organization of the modular Petri Nets is of the highest importance, i.e., strict adherence to proven architectural principles, such as modularization, correct partitioning (respecting cohesion and coherence), loose coupling, and separation of concerns (e.g., Furrer, 2019; Platzer, 2018). This design of the adequate structure is independent of the formal specification of the system and must be carried out by very experienced software architects. Transformation path A requires a strong architecture governance in the IT organization (e.g., Murer, 2014 / Bell, 2023). Once the Petri Modules/IMC structural architecture has been defined, the states and transitions that are to be encapsulated by each Petri Module are selected (Figure 7). Once all states, transitions, and quality properties are transferred from the timed transition system to the timed Petri Module system, the duty of transformation path 1 is completed. Today, transformation Path 1 is state-of-the-art in methodology and architecture knowledge.

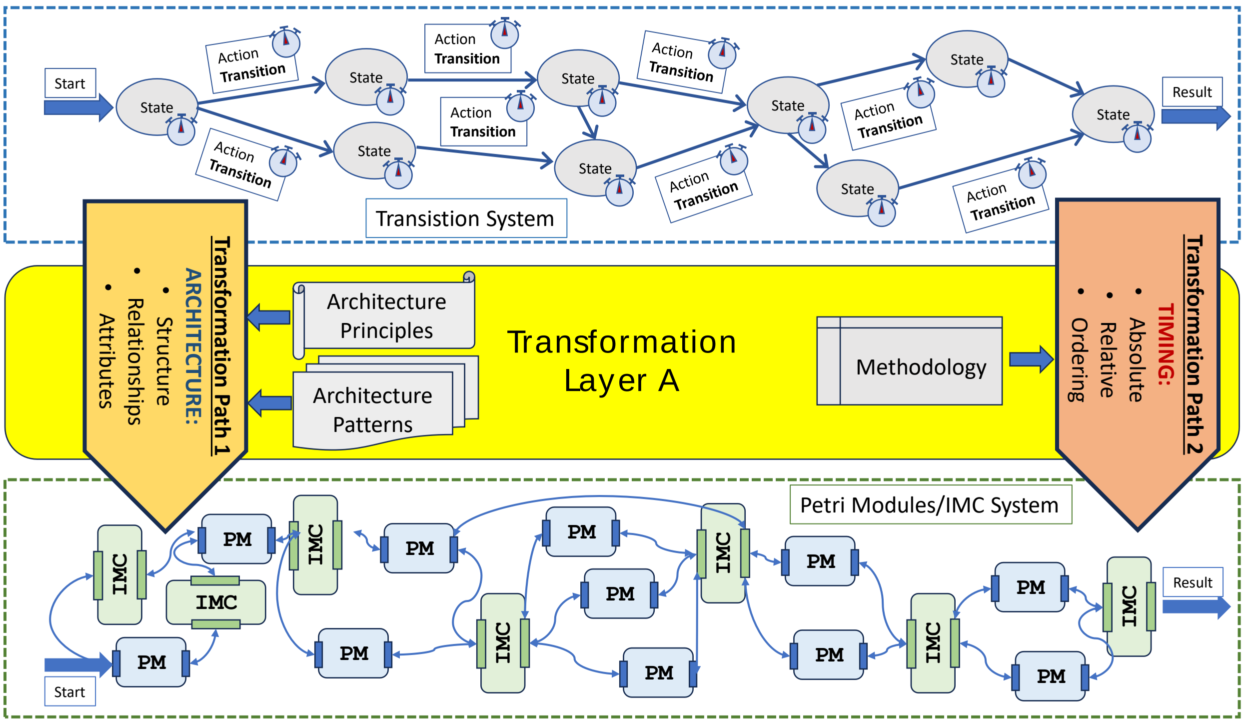


Figure 7: Elements of the Transformation Layer A

Transformation Path 2 (Timing, Figure 7):

Timed transition systems (e.g., Furia et al., 2012, chapters 7.3 & 7.4 / Henzinger et al., 1991 / Hale et al., 1994) and timed Petri Nets have different formal notations for time representation (e.g., Furia et al., 2012, chapter 8 / Wang, 1998 / Penczek et al., 2006). These different notations have differing expressiveness, and suitable notations must be selected for this application.

The transformation path 2 transcribes the transition system timing information to the Petri Net timing information (e.g., Best et al., 1998), including all constraints. Promising initial work has been done on such transformations (e.g., Khomenko et al., 2022 / Huang et al., 2021), but more consolidating research is needed for this transformation path, focussed on the proposed architecture.

Transformation Layer B

Context: The responsibility of the transformation layer B is to select one or more Petri Modules and use them to form a task (Figure 8). This includes correctly transforming not only the functionality and data, but also the timing and the quality properties.

The transformation layer B has two transformation paths (Figure 8):

Transformation path 3 (Architecture):

Transformation path 3 selects one or several coherent Petri Modules, allocates them to specific tasks, and uses the IMCs to define the relationships from task to task and from task to the environment. While the adequate architecture (structure, relationships) has already been defined by transformation path 1, the transfer of functionality/data/relationships/quality attributes from the Petri Module system to the task universe by the transformation path 3 must at least preserve – preferably improve – the quality of the software architecture. This means, again, strict adherence to proven architectural principles and patterns, such as modularization, correct partitioning (respecting cohesion and coherence), loose coupling, and separation of concerns, etc. (e.g., Furrer, 2019 / Richards et al., 2025 / Martin, 2017 / Cervantes, 2024 / Khononov, 2025 / Fettke et al., 2022). Once all Petri Modules/IMC are transferred to the task structure, the duty of transformation path 3 is completed. Today, transformation Path 3 is state-of-the-art in terms of both methodology and architecture knowledge.

Transformation path 4 (Timing):

Transformation path 4 transfers the timing specifications from the Petri Net module system to the task universe, i.e., to the implementation level. Timing in Petri Nets is introduced associated with places, transitions, or both. Some work has been done on software implementations of timed Petri Nets (e.g., Girault et al., 2010 (Chapters 20 & 21 / Ferscha, 1994 / Barad, 2016 / Moreno et al., 2006 / Andrezejwski, 2001). However, neither approach is sufficient for the application to the transformation path 4. Therefore, transformation Path 4 needs more research, specifically directed to the proposed architecture.

… and one feedback path (Timing adjustments): Timing Feedback

The applications prescribe the timing requirements for the system (Processes in Figure 1). At the moment of timing specification, there is no guarantee that their successful implementation will be feasible (e.g., Klemm et al., 2021 / Philippou et al., 2007). The following obstacles may appear:

- Some tasks may have an unexpectedly large WCET (Worst Case Execution Time);
- The task system is not schedulable (TTA);
- The physical communications channel’s transmission times negatively impact timing;
- The system does not provide sufficient resources to handle concurrency and latency;
- Correct error and fault handling require more resources than expected;
- etc.

If the timing can not be implemented in the real CPS, three resorts are possible:

I. Weaken the initial timing requirements (if the applications/processes allow it);
II. Try to modify the architecture (Structure, relationships);
III. Provide more implementation resources.

Once the complete system of timed Petri Net modules has been transferred into tasks and their relationships, and the feasibility of the implementation has been assured, the mission of transformation layer B is complete.

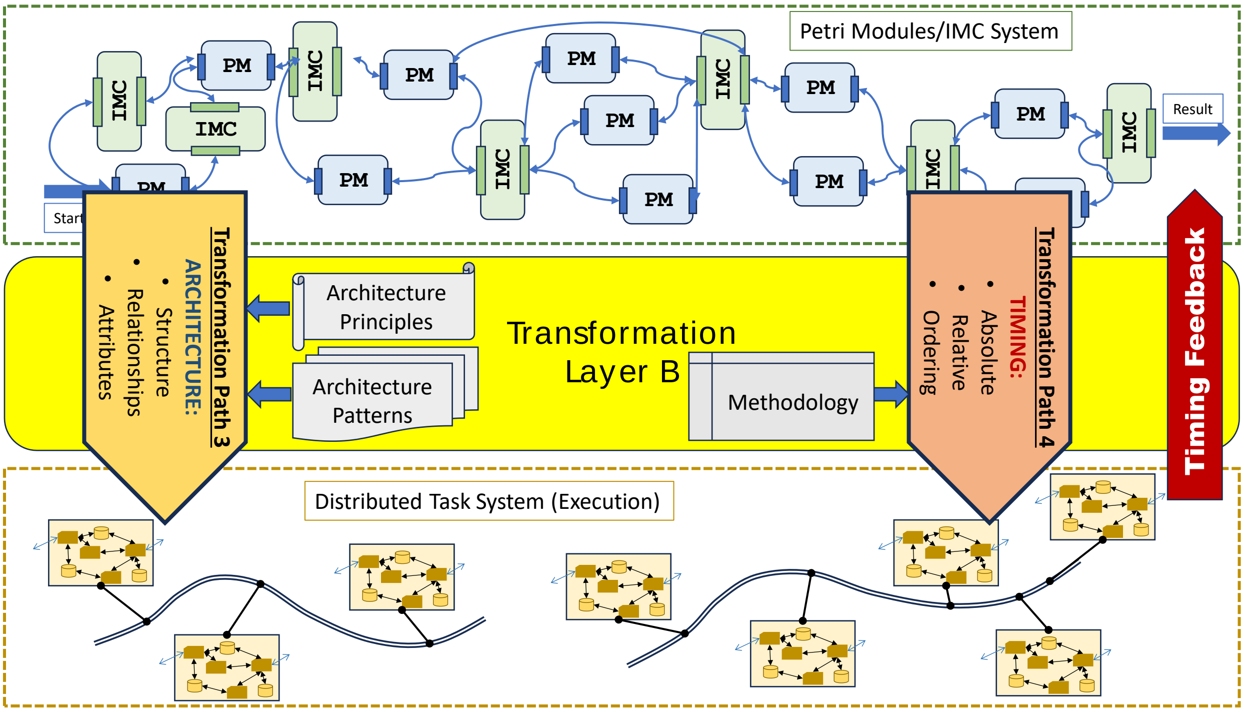


Figure 8: Elements of the Transformation Layer B

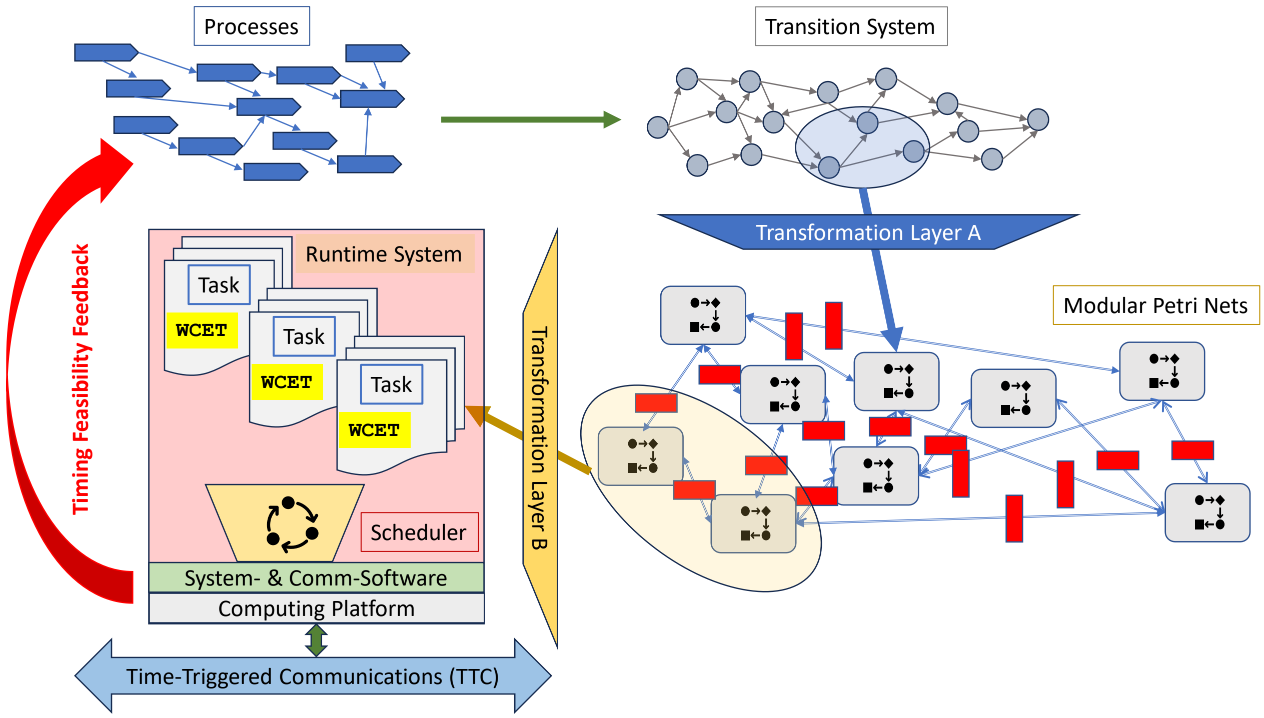


Figure 9: Transformation Layers and Runtime System

Time-Triggered Protocol (TTP) - Time-Triggered Architecture (TTA)

The time-triggered architecture (TTA) defines a fault-tolerant execution platform for large, distributed, embedded real-time systems in mission- and safety-critical cyber-physical applications, such as avionics (e.g., Fuhrmann et al., 2006). It is based on the time-triggered model of computation (Kopetz, 1998 / Kopetz, 2017) and introduces the paradigm of time-triggered communications (TTC, e.g., Kopetz et al., 2003 / Obermaisser, 2012 / Kopetz, 2022/ /Maier et al., 2002/ Rushby, 2005/Buttazzo, 2023). The basic concepts of TTA are shown in Figure 10. Note that the time-triggered communication (TTC) is a paradigm for electronic information exchange (as opposed to the event-triggered communications), the time-triggered protocol (TTP) is the implementation, and the time-triggered architecture (TTA) includes in addition system components, such as scheduler, redundant communication channel, global time synchronization, etc. (Figure 10).

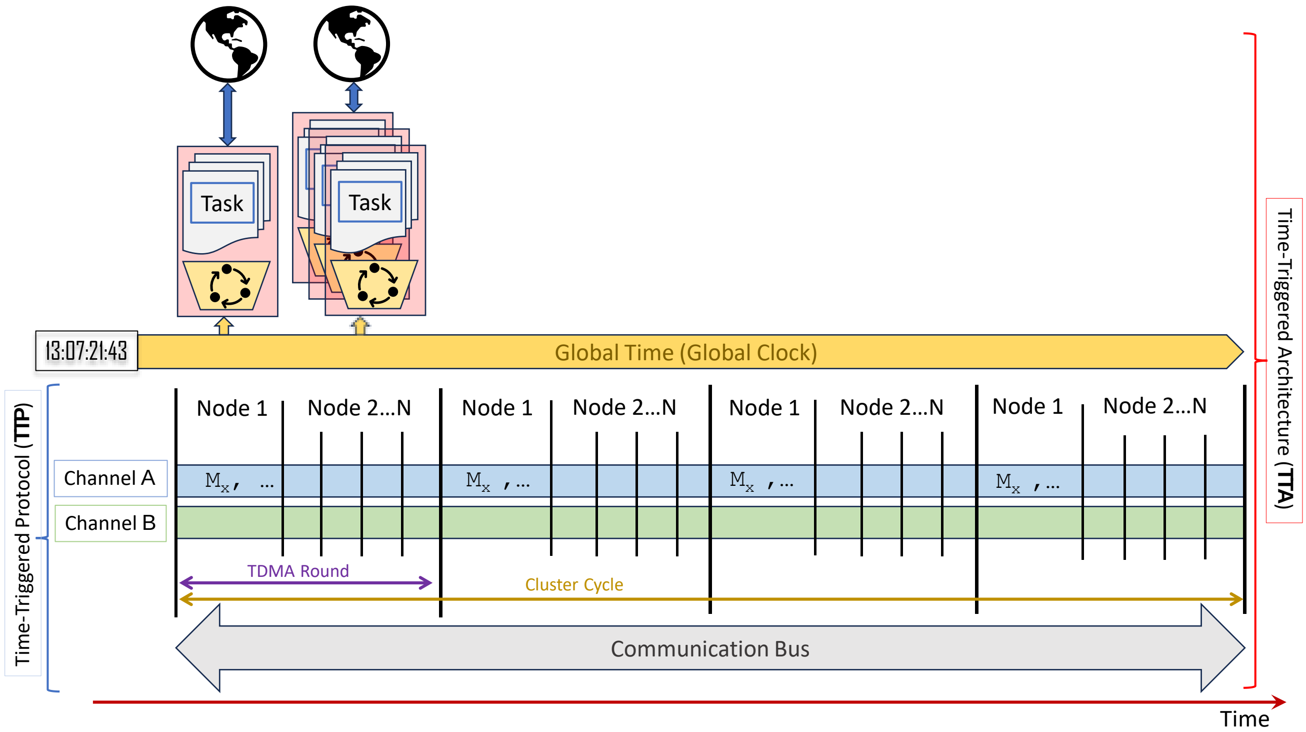


Figure 10: Time-Triggered Architecture

The Figure 10 introduces the following elements (From lowest to highest):

(1) A redundant communication bus that allows the exchange of messages. Initially, a TDMA (Time Division Multiple Access)-scheme was used in the TTP. Later – forced by industry standardization – TTP was implemented on top of more communication schemes, such as CAN (Führer et al., 2000), Ethernet (Kopetz et al., 2005), and FlexRay (Shaw et al., 2008);

(2) Two time-triggered protocols (TTP), managing the exchange of messages between the N nodes in the network, are implemented on top of the two communication channels, providing the necessary redundancy for safe operation. TTP provides fault-tolerant message transport with a fixed schedule at known times and minimal jitter by employing a TDMA (Time-Division Multiple Access) strategy;

(3) A protocol to establish a global, synchronized time in all the nodes. TTA provides system-wide, fault-tolerant, and distributed clock synchronization, establishing a global time base without relying on a central time server.

(4) The runtime systems in each node, i.e., a set of tasks governed by a scheduler.

(5) Several algorithms for system functions (Obermaisser et al., 2012, Chapter 4):

i. Clock synchronization,
ii. Startup and Restart,
iii. Diagnostic Services,
iv. Error Detection and Fault Isolation,
v. Configuration Service,
vi. Schedule Generation and Schedulability Analysis

(6) The interfaces for the interaction of the tasks with the physical world (Sensors, Actuators).

The Time-Triggered Protocol is a deterministic, verifiable, well-analyzed message exchange scheme for fault-tolerant, distributed systems (e.g., Rushby, 2002). Therefore, it forms a predictable foundation for the execution platform in Figure 6.

Worst-Case Execution Time (WCET)

Each program (= a piece of code) has a worst-case execution time (WCET, e.g., Lokuciejewski, 2011). The worst-case execution time (WCET) of a program is the maximum amount of time the program could take to execute on a specific execution platform, i.e., the longest path through the program. Unfortunately, the WCET determination corresponds to the halting problem and is therefore not generally solvable. Estimation methods, such as simulation and code analysis (e.g., Franke, 2016 / Ferdinand et al., 2004), must be used to obtain valuable results. For time-sensitive software, the WCET of each program/module/task must be determined with sufficient accuracy (e.g., Wolf, 2002).

Runtime System and Task Scheduling

The resulting runtime system is shown in Figure 9. It consists of a set of tasks, system- and communications software, a computing platform (today often a cached, multicore CPU), the

TTC bus, and a task scheduler. The scheduler orchestrates the sequence of execution of the tasks in the distributed nodes of the system.

Except for the scheduling, all elements of the conceptual architecture in Figure 6 have been chosen due to the predictability and verifiability of their correct timing behavior. Scheduling, preemption, and resource sharing may cause timing uncertainties and must be analyzed and implemented very carefully. A rich literature related to building, verifying, and operating predictable, hard real-time computing platforms exists (e.g., Buttazzo, 2024 / Gliwa, 2022 / Obermaisser, 2012 [Chapter 15] / Ayman et al., 2009 / Antolak et al., 2023). There is no space to handle this topic, only to raise awareness.

Cyber-physical systems need global time, i.e., a system-wide, precise, and synchronized common physical time scale in all elements of the CPS (Shrivastava et al., 2016 / Broman et al., 2013 / Rajeev et al., 2012). In the conceptual architecture of Figure 2, the Time-Triggered Architecture provides the global clock (Figure 10 / Obermaisser, 2011, Chapter 4).

Mixed-Criticality Systems

Many CPSs are “mixed-criticality systems”, i.e., they contain time-sensitive processes/parts and non-time-sensitive processes/parts. The system design must be based on solid partitioning and loose coupling between the two criticality regions.

Timing Verification

The final truth of timing correctness lies in the runtime system (Lowest layer in Figure 6). Only if the runtime system strictly adheres to all timing specifications in all operating conditions can it be qualified as safe. The strong formalism and model-checking capabilities of the 3 top layers in Figure 6 ensure high confidence in the system timing conformance with the specifications because of the formal verification. Process algebras, transition systems, and Petri Nets allow the verification of their timing properties (e.g., Becker, 2020 / Willemse, 2003 / Camargo, 1998 / Corradini et al., 1999 / Philippou et al., 2007 / Penczek et al., 2006 / Wolf, 2002).

Timing verification on the lowest layer in Figure 6 (Runtime system) requires measurements, tracing, statistics, analysis, and assessment (e.g., Rohr, 2015 / Becker, 2020). Runtime verification, especially for the timing, is a challenging task but sufficiently researched (e.g., Colombo et al., 2022).

Real-Time Calculus (RTC)

A promising development for formalizing the timing behavior and formal verification of the runtime system is the real-time calculus RTC (e.g., Guan, 2018 / Thiele et al., 2000 / Two Examples: Chokshi, 2010 / Bazzal et al., 2020). The key concept in RTC is the Greedy Processing Component (GPC, Figure 11). The GPC accepts input events, launches the appropriate processing, and outputs the processed event stream. The event streams are formalized by arrival curves based on the number of events arriving at an interval (one for the lower bound, the other for the upper bound). The resources consumed to process the input events are also formalized by service curves based on the amount of resources consumed in an interval Δ, one for the lower bound, the other for the upper bound.

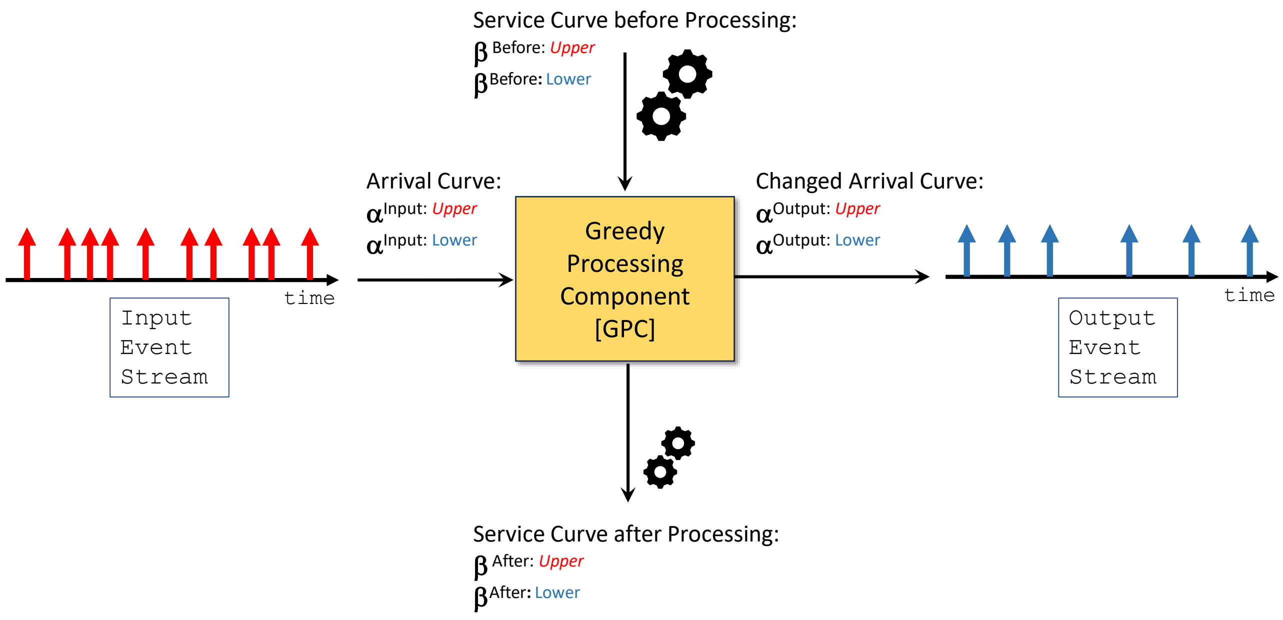


Figure 11: RTC Key Concept – Greedy Processing Component (GPC)

For the arrival and service curves, operators are defined to build compositions of GPCs and thus describe systems of arbitrary complexity. The benefits of the RTC include the formalism for determining bounds for execution, communication, queues, and buffer sizes. Additionally, the schedulability of multitasking software systems can be determined using Real-Time Calculus (RTC).

Runtime Monitoring

As a last defense against timing violations, runtime monitoring can be used. Whereas runtime verification aims to check specific parameters of the program execution, such as the execution times of a set of tasks, runtime monitoring supervises the system in order to detect anomalous or dangerous behavior. If anomalous behavior is detected, the system may automatically take protective actions, thus trying to avoid safety accidents or security incidents. Machine learning algorithms are often used for anomaly detection. (e.g., Furrer, 2023).

Results

Strict adherence to timing requirements is a crucial precondition for the safety of cyber-physical systems. Therefore, the software controlling the CPS becomes time-sensitive. The conceptual system architecture is the foundation for the assurance of timing requirements in a CPS. Only an adequate system architecture allows the formal specification, verification, modeling, and implementation of timing requirements on all levels and for all process steps.

This paper proposes a novel timing-aware architecture composed of well-known technologies: process algebra for modeling transition processes, Petri Nets for implementation, and time-triggered communications as the execution platform. The timing-aware 4-layer architecture is presented as a conceptual 4-layer model. From this model, many research topics follow.

Open Questions and Future Work

- Develop a complete and consistent metamodel to ensure the conceptual integrity of all layers in Figure 2 (e.g., Gonzalez-Perez et al., 2008)
- Choose and agree on a semantic and notation for a suitably timed process algebra. Codify it as an industry standard;
- Choose and agree on a semantic and notation for timed transition systems. Propose it as an industry standard;

- Choose and agree on a semantic and notation for a timed Petri Nets (Preferably based on Davidrajuh, 2021). Propose it as an industry standard;
- Develop, discuss, and document a modeling methodology for systems based on Figure 6 (Metamodel, notation, semantics, graphical representation, etc.);
- Define a methodology, principles, and metrics for the transformation layer A;
- Define a methodology, principles, and metrics for the transformation layer B;
- Integrate the formalism of real-time calculus (RTC) into the architecture of Figure 6;
- Investigate the applicability of the (possibly extended) conceptual architecture of Figure 2 to continuous and hybrid cyber-physical systems (e.g., David et al., 2010 / Gu et al., 2005 / Bera et al., 2014 / David et al., 2010);
- Demonstrate the capability of the conceptual architecture (Figure 2) for closed-loop CPS (e.g., Pasandideh et al., 2023 / Núñez-Alvarez et al., 2023);
- Does the conceptual architecture (Figure 2) have the capability to generate the most efficient solution regarding resources? (Rodriguez, 2013 / Jarabo, 2024 / Shi, 2023);
- Is adding more resources until timing constraints can be satisfied always feasible?

Conclusions

For mission-critical cyber-physical systems, the correct specification, implementation, and execution of complete timing specifications is a *correctness property* rather than a *quality attribute*. To answer this challenge, the underlying system architecture must provide formal, verifiable, and complete timing constructs on all levels. This paper proposes a novel, four-layer architecture with sufficient formalism based on established technologies to handle and verify timing in a Cyber-Physical System (CPS).

Declarations

The author is the only contributor to this paper.
The author has no conflicts of interest.
This research received no specific grant from any funding agency, commercial or not-for-profit sectors.
Data availability does not apply to this article as no new data were created or analyzed in this study.
Ethical approval and consent are not relevant to this article type.

Acknowledgments

First and foremost, I would like to extend my sincere thanks to my colleagues and students at the Computer Science Faculty of the Technical University of Dresden, Germany. In my ten years of teaching, they enabled me to gain extensive knowledge in many new areas. Next, I would like to extend my sincere thanks to all the authors (listed in the references) who provided the knowledge base for this paper. Special thanks are due to Prof. Dr. Hermann Kopetz (Technical University of Vienna) for numerous discussions on real-time systems and their architecture. Finally, I thank Mónica Moniz and Ellie Pilat (from Cambridge University Press, UK) and Jim Woodcock (University of York, UK) for their valuable support during the preparation of this paper, as well as the two reviewers: (1) Associate Professor Arvind Easwaran, NTU, Singapore (named with permission), and (2) Professor Partha Roop, University of Auckland, New Zealand (also named with permission) who significantly improved the content and quality of this paper.

Connections Reference

Lee E. A., Woodcock J. (2023): Time-Sensitive Software. *Research Directions, Cyber-Physical Systems, Vol 1. Cambridge University Press, Cambridge, UK*. https://doi.org/10.1017/cbp.2023.1

General References

Abdellatif, T., Combaz, J., Sifakis, J. (2010): Model-based implementation of real-time applications. *EMSOFT ’10: Proceedings of the tenth ACM international conference on Embedded software, pp. 229 – 238*. https://doi.org/10.1145/1879021.1879052

Aceto, L. (2003): Some of my Favourite Results in Classic Process Algebra. *BRICS Notes Series NS-03-2. University of Aarhus, Aarhus, Denmark*. ISSN 0909-3206. Downloadable from: https://www.brics.dk/NS/03/2/BRICS-NS-03-2.pdf

Ahmadi, A., Cherifi C., Cheutet, V., Ouzrot, Y., (2021): A Review of CPS 5 Components Architecture for Manufacturing Based on Standards. *11th IEEE International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2017), Colombo, Sri Lanka*. Downloadable from: https://www.researchgate.net/publication/224264704_Design_and_architectures_for_dependable_embedded_systems

Aldini A., Bernardo M., Corradini F. (2009): A Process Algebraic Approach to Software Architecture Design. *Springer Verlag, London, UK*. ISBN 978-1-84800-222-7. https://doi.org/10.1007/978-1-84800-223-4

Andrezejwski, G. (2001): Timed Petri Nets for Software Applications. *The International Workshop on Discrete-Event System Design, DESDes'01, June 27-29, 2001, Przytok near Zielona Gora, Poland*. Downloadable from: http://www.iie.uz.zgora.pl/iie_archiwum/desdes01/files/ref/I-9.pdf

Antolak, E., Pulka, A. (2023): Validation of Task Scheduling Techniques in Multithread Time Predictable Systems. *IEEE Access, New York, NY, USA*. https://doi.org/10.1109/ACCESS.2023.3275437. Downloadable from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10122958 (Open Acess)

Ayman K. G. Gendy, A.K.G. (2009): Techniques for scheduling time-triggered resource-constrained embedded systems. Embedded Systems Laboratory. *PhD Thesis, Department of Engineering, University of Leicester, Leicester, UK*. Downloadable from: https://figshare.le.ac.uk/articles/thesis/Techniques_for_scheduling_time-triggered_resource-constrained_embedded_systems/10092473?file=18194459

Badouel, E., Bernardinello, L., Darondeau, P. (2015): Synthesis of P/T-Nets from Finite Initialized Transition Systems. *Chapter 7 in Badouel, E., Bernardinello, L., Darondeau, P. (2015): Petri Net Synthesis. Springer Verlag, Berlin, Germany*. ISBN 978-3-662-47966-7. https://doi.org/10.1007/978-3-662-47967-4

Baeten, J.C.M., Middelburg C.A. (2001): Process Algebra with Timing - Real-Time and Discrete Time. CHAPTER 10 in: Bergstra J.A., Ponse A., Smolka S.A. (Editors, 2001): Handbook of Process Algebra. *Elsevier Science B.V., Amsterdam, The Nederlands, Pages 627-684*. ISBN 978-0-444-82830-9.

Baeten J.C.M., Middelburg C.A. (2002): Process Algebra with Timing. *Springer Verlag, Berlin, Germany.* ISBN 978-3-540-43447-4

Baeten, J.C.M. (2005): A Brief History of Process Algebra. Theoretical Computer Science Journal, 335(2-3), p. 131-146. 2025 Elsevier B.V., Amsterdam, The Netherlands. https://doi.org/10.1016/j.tcs.2004.07.036. Downloadable from: https://www.researchgate.net/publication/220150532_A_brief_history_of_process_algebra

Barad, M. (2016): Petri Nets—A Versatile Modeling Structure. *Applied Mathematics, Vol.7 No.9, May 2016*. http://dx.doi.org/10.4236/am.2016.79074. Downloadable from: https://www.scirp.org/pdf/AM_2016052613411594.pdf

Bazzal, M., Krawczyk, L., Govindarajan, R.P., Wolff, C. (2020): Timing Analysis of Car-to-Car Communication Systems Using Real-Time Calculus - A Case Study. *The 5th IEEE International Symposium on Smart and Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems, 17-18 September 2020, Dortmund, Germany*. Downloadable from: https://www.researchgate.net/figure/A-Greedy-Processing-Component-GPC-element-transforms-the-input-pairs-of-arrival-and_fig2_347867475

Becker, M., 2020: Towards Source-Level Timing Analysis of Embedded Software Using Functional Verification Methods. *PhD Thesis, Fakultat fur Elektrotechnik und Informationstechnik der Technischen Universitat Munchen, Munich Germany*. Downloadable from: https://mediatum.ub.tum.de/doc/1506241/1506241.pdf

Bell, M., 2023: Software Architect. *John Wiley & Sons Inc., Hoboken, NJ, USA*. ISBN 978-1-119-82097-0.

Bellini, R., Mattolini, P., Nesi, P. (2000): Temporal Logics for Real-Time System Specification. *ACM Computing Surveys (CSUR), Volume 32, Issue 1, pp. 12 – 42.* https://doi.org/10.1145/349194.349197. Downloadable from: https://dl.acm.org/doi/10.1145/349194.349197.

Bera, D., Van Hee, K.M., Nijmeijer, H. (2014): Modeling hybrid systems with Petri nets. *4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH 2014 - Vienna, Austria, 28.-30.4.2014*. Downloadable from: https://research.tue.nl/en/publications/modeling-hybrid-systems-with-petri-nets.

Bergstra J.A., Klop J.W. (1984): Process Algebra for Synchronous Communication. *Information and Control, Elsevier, Amsterdam, NL*. Downloadable from: https://ir.cwi.nl/pub/1836/1836D.pdf

Best, E., Devillers, R., Koutny, M. (1998). Petri nets, process algebras, and concurrent programming languages. *In: Reisig, W., Rozenberg, G. (editors) Lectures on Petri Nets II: Applications. ACPN 1996. Lecture Notes in Computer Science, vol 1492. Springer, Berlin, Heidelberg*. https://doi.org/10.1007/3-540-65307-4_46. Downloadable from:

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7f08fe879a322db3b04dec51a06c9417d5e02ba4

Best, E., Devillers, R. (2024): Synthesis of Petri Nets from Labelled Transition Systems. *Chapter 11 in Best, E., Devillers, R. (2024): Petri Net Primer - A Compendium on the Core Model, Analysis, and Synthesis. Birkhäuser Verlag, Basle, Switzerland*. ISBN 978-3-031-48277-9. https://doi.org/10.1007/978-3-031-48278-6_11

Bolc, L., Szalas, A. (Editors, 2019 / initially published in 1995): Time & Logic - A Computational Approach. *Routledge Revival, Milton Park, Abingdon, UK*. ISBN 978-0-367-33657-8.

Börger, E., Stärk, R. (2013): Abstract State Machines - A Method for High-Level System Design and Analysis. *Springer Verlag, Berlin. Germany*. ISBN 978-3642621161.

Bolognesi, T., Brinksma, E. (1968): Introduction to the ISO Specification Language LOTOS. *Technical Report CNUCE-C.N.R., Pisa, Italy & University of Twente, Enschede, The Netherlands*. Downloadable from: https://cadp.inria.fr/ftp/publications/others/Bolognesi-Brinksma-87.pdf

Broman, D., Derler, P., Eidson, J.C. (2013): Temporal Issues in Cyber-Physical Systems. *Journal of the Indian Institute of Science, Vol. 93, Nr 3, July/September 2013. Bangalore, India.* Downloadable from: https://journal.iisc.ac.in/index.php/iisc/article/view/1686

Buckl, C., Gaponova, I., Geisinger, M., Knoll, A., Lee, E.A. (2010): Model-based specification of timing requirements. *EMSOFT '10, Proceedings of the tenth ACM international conference on Embedded software, pp. 239 – 248. Scottsdale, Arizona, USA, October 24-29, 2010.* https://doi.org/10.1145/1879021.1879053. Available at: https://dl.acm.org/doi/10.1145/1879021

Buttazzo G. (2023): Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications. 4th edition. *Springer Nature Switzerland, Cham, Switzerland*. ISBN 978-3-031-45409-7. https://doi.org/10.1007/978-3-031-45410-3

Camargo, M. (1998): Formal Specification, Verification, and Simulation of Time-Dependent Systems - A Timed Process Algebra Approach. *Electronic Notes in Theoretical Computer Science. Published by Elsevier Science BV, Amsterdam, Netherlands.* Downloadable from:

https://www.academia.edu/123683998/Formal_Specification_Verification_and_Simulation_of_Time_Dependent_Systems_a_Timed_Process_Algebra_Approach

Cervantes, H., Kazman, R. (2024): Designing Software Architectures - A Practical Approach. *Pearson Education, Addison-Wesley, Boston, USA.* ISBN 978-0-138-10802-1

Chao W.S. (2015): A Process Algebra For Systems Architecture - The Structure-Behavior Coalescence Approach. *CreateSpace Independent Publishing Platform*. ISBN 978-1-517-25861-0

Chokshi, D.B., Bhaduri, P. (2010): Performance analysis of FlexRay-based systems using real-time calculus. *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10, Sierre, Switzerland.* ISBN 978-1-60558-638-0/10/03. Downloadable from: https://www.academia.edu/13201410/Performance_analysis_of_FlexRay_based_systems_using_real_time_calculus_revisited

Colombo, C., Pace, G.J. (2022): Runtime Verification - A Hands-On Approach in Java. *Springer Nature Switzerland AG, Cham, Switzerland.* ISBN 978-3-031-09266-4. https://doi.org/10.1007/978-3-031-09268-8

Corradini, F., D'Ortenzio, D., Inverardi, P. (1999): On the Relationship among four Timed Process Algebras. *Fundamenta Informaticae, 38 (1999), Nr. 4, pp. 377-395. IOS Press, IOS Press, Amsterdam, Netherlands*. Available from: https://content.iospress.com/articles/fundamenta-informaticae/fi38-4-03Cortadella, J., Lavagno, L., Kishinevsky, M., Yakovlev, A. (1995): Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers, vol. 47, no. 8, pp. 859-882, Aug. 1998.* https://ieeexplore.ieee.org/document/707587. Downloadable from: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bc1c6bf9a14e8683a0e3f02433d265c1a87d9fb9

David, R., Alla, H. (2010): Discrete, Continuous, and Hybrid Petri Nets. *Springer Verlag, Berlin, Germany.* 2nd edition. ISBN 978-3-642-10668-2. https://doi.org/10.1007/978-3-642-10669-9

Davidrajuh R. (2021): Petri Nets for Modeling of Large Discrete Systems. *Springer Nature Singapore, Singapore*. ISBN 978-981-16-5202-8. https://doi.org/10.1007/978-981-16-5203-5

Debouk, R. (2019): Overview of the Second Edition of ISO 26262: Functional Safety – Road Vehicles. *Journal of System Safety, Saint Paul, MS, USA, Vol. 55, Nr. 1*. https://doi.org/10.56094/jss.v55i1.55

Demri S., Goranko V., Lange M. (2016): Temporal Logics in Computer Science - Finite-State Systems. *Cambridge Tracts in Theoretical Computer Science, Vol. 58. Cambridge University Press, Cambridge, UK.* ISBN 978-1-107-02836-4

DeNicola R. (2011): A gentle introduction to Process Algebras. *IMT- Institute for Advanced Studies, Lucca, Italy*. Downloadable from: https://www.pst.ifi.lmu.de/Lehre/fruhere-semester/sose-2013/formale-spezifikation-und-verifikation/intro-to-pa.pdf

Devillers, R., Tredup, R. (2022): Some Basic Techniques allowing Petri Net Synthesis - Complexity and Algorithmic Issues. *Fundamenta Informaticae, Volume 187, Issues 2-4: Petri Nets. Villeurbanne, France.* Downloadable from: https://doi.org/10.48550/arXiv.2112.03605

Ferdinand, C., Heckmann, R. (2004): aiT – Worst-Case Execution Time Prediction by Static Program Analysis. *In: Building the Information Society, IFIP 18th World Computer Congress Topical Sessions, 22–27 August 2004, Toulouse, France, pp 377–383. Springer New York, NY, USA*. https://doi.org/10.1007/b98986. Downloadable from: https://link.springer.com/content/pdf/10.1007/978-1-4020-8157-6_29.pdf (Open Access)

Ferscha, A. (1994): Concurrent execution of timed Petri nets. *Proceedings of Winter Simulation Conference, Lake Buena Vista, FL, USA, 1994, pp. 229-236*. https://doi.org/10.1109/WSC.1994.717133

Fettke, P., Reisig, W. (2022): Modularization, Composition, and Hierarchization of Petri Nets with Heraklit. *German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany*. https://doi.org/10.48550/arXiv.2202.01830. Downloadable from: https://arxiv.org/abs/2202.01830.

Fokkink W. (1999): Introduction to Process Algebra. *Springer Verlag, Berlin, Germany*. ISBN 978-3-540-66579-3

Franke B. (2016): Embedded Systems - Lecture 11: Worst-Case Execution Time. *The University of Edinburgh, Edinburgh, Scotland.* Downloadable from: https://www.inf.ed.ac.uk/teaching/courses/es/PDFs/lecture_11.pdf

Führer, T., Müller, B., Dieterle, W., Hartwich, F. Hugel, R., Walther, M. (2000): Time-Triggered Communication on CAN (TTCAN), Draft for TC 22/SC3/WG1/TF6 (ISO 11898-4). *Robert Bosch Gmbh, Suttgart, Germany*. Downloadable from: https://www.jstor.org/stable/44718317

Fuhrmann, H., von Hanxleden, R., Rennhack, J., Koch, J. (2006): Model-Based System Design of Time-Triggered Architectures - Avionics Case Study. *2006 ieee/aiaa, 25th Digital Avionics Systems Conference. Portland, OR, USA*. https://doi.org/10.1109/DASC.2006.313745

Furia C.A., Mandrioli D., Morzenti D., Ross M. (2012): Modeling Time in Computing. *Springer Science & Business Media, Berlin, Germany.* ISBN 978-3-642-32331-7. https://doi.org/10.1007/978-3-642-32332-4

Furrer F.J. (2019): Future-Proof Software-Systems - A Sustainable Evolution Strategy. *Springer Vieweg Fachmedien, Wiesbaden, Germany*. ISBN 978-3-658-19937-1. https://doi.org/10.1007/978-3-658-19938-8

Furrer F.J. (2022): Safety and Security of Cyber-Physical Systems - Engineering dependable Software using Principle-based Development. *Springer Vieweg Fachmedien, Wiesbaden, Germany*. ISBN 978-3-658-37181-4. https://doi.org/10.1007/978-3-658-37182-1

Furrer F.J. (2023): Safe and secure system architectures for cyber-physical systems. *Informatik Spektrum 46, 96–103, Open access*. https://doi.org/10.1007/s00287-023-01533-z

Gaska, T., Watkins, C., Chen, Y. (2015): Integrated Modular Avionics — Past, Present, and Future. *IEEE Aerospace and Electronic Systems Magazine, 30 (9), September 2015, pp. 12-23*. https://doi.org/10.1109/MAES.2015.150014. Downloadable from: https://www.researchgate.net/publication/284113137_Integrated_Modular_Avionics_-_Past_present_and_future

Girault C., Valk R. (2010): Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications. *Springer Verlag, Berlin, Germany*. ISBN 978-3-642-07447-9.

Gliwa, P. (2022)E: Embedded Software Timing - Methodology, Analysis, and Practical Tips with a Focus on Automotive. *Springer Nature Switzerland AG, Cham, Switzerland*. ISBN 978-3-03-064146-7. https://doi.org/10.1007/978-3-030-64144-3

Goltz, U. (1990): CCS and Petri Nets. *In: Guessarian, I. (editor): Semantics of Systems of Concurrent Processes. LITP 1990. Lecture Notes in Computer Science, vol 469. Springer, Berlin, Germany*. ISBN 978-3-540-53479-2. https://doi.org/10.1007/3-540-53479-2_14

Gonzalez-Perez, C., Henderson-Sellers, B. (2008): Metamodelling for Software Engineering. *John Wiley & Sons, Chichester, UK*. ISBN 978-0-470-03036-3

Gorrieri R., Versari C. (2015): Introduction to Concurrency Theory - Transition Systems and CCS. *Springer International Publishing Switzerland, Cham, Switzerland*. ISBN 978-3-319-36638-8. https://doi.org/10.1007/978-3-319-21491-7

Griffor, E., Greer, C., Wollman, D. and Burns, M. (2017): Framework for Cyber-Physical Systems. Volume 1, Overview. *Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, USA*. https://doi.org/10.6028/NIST.SP.1500-201. Downloadable from: https://www.nist.gov/publications/framework-cyber-physical-systems-volume-1-overview

Gu, T., Dong, R. (2005): A novel continuous model to approximate time Petri nets - Modelling and Analysis. *International Journal of Applied Mathematics and Computer Science, Vol 15, Nr. 1, pp. 141–150. University of Zielona Góra, Poland*. Available at: https://www.amcs.uz.zgora.pl/?action=paper&paper=234

Guan, N. (2018): Techniques for Building Timing-Predictable Embedded Systems. *Springer International Publishing Switzerland, Cham, Switzerland*. ISBN 978-3-319-80089-9. https://doi.org/10.1007/978-3-319-27198-9

Hale, R.W.S., Cardell-Oliver, R.M., Herbert J.M.J. (1994): Real-Time Safety Critical Systems. https://doi.org/10.1016/B978-0-444-89901-9.50013-6. *Chapter 4 in: Towards Verified Systems, Volume 2, 1994, Pages 71-90. Elsevier, Amsterdam, The Netherlands*. https://doi.org/10.1016/B978-0-444-89901-9.50013-6. Downloadable from: https://www.comp.nus.edu.sg/~cs5270/Notes/chapt4.pdf

Hawking S. (2015): The Illustrated Brief History of Time. *Bantam Books, New York, NY, USA*. ISBN 978-0-593-07718-4

Henzinger, T., Manna, Z., Pnueli, A. (1991): Timed Transition Systems. *Pages 226-251 in: Real-Time - Theory in Practice, Proceedings of the REX Workshop, Mook, The Netherlands,* *June 3-7, 1991. Springer-Verlag, Berlin, Germany. LNCS, volume 600*. https://doi.org/10.1007/BFb0031984.

Huang, H., Jiao, L., Cheung, T.Y., Wak, W.M. (2012): Property-preserving Petri Net Process Algebra in Software Engineering. *World Scientific Publishing, Singapore, Singapore*. ISBN 978-9-81-432428-1

Klemm, M., Cownie, J. (2021): High-Performance Parallel Runtimes - Design and Implementation. *Walter de Gruyter Gmbh, Berlin, Germany*. ISBN 978-3-11-063268-2

Hoare C.A.R. (1985): Communicating Sequential Processes. *Prentice-Hall International, Englewood Cliffs, USA.* ISBN 978-0-131-53289-2. Downloadable from: https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf

ISO (2001) International Organization for Standardization: ISO/IEC 15437:2001(en), Information technology — Enhancements to LOTOS (E-LOTOS). *Joint Technical Committee ISO/IEC JTC 1, Information Technology, Subcommittee SC 7, Software Engineering. Geneva, Switzerland*. Available at: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:15437:ed-1:v1:en

Jarabo, J.I.R., Gómez-Martínez1, E., Kallwies, H., Haustein, M., Leucker, M., Stolz, V., Stünkel, P. (2024): Runtime Verification of Timed Petri Nets. *PNSE'24, International Workshop on Petri Nets and Software Engineering, Geneva, Switzerland, 2024*. Downloadable from: https://ceur-ws.org/Vol-3730/paper07.pdf

Khomenko, V., Koutny, M., Yakovlev, A. (2022): Slimming down Petri Boxes - Compact Petri Net Models of Control Flows. *In 33rd International Conference on Concurrency Theory (CONCUR 2022). Leibniz International Proceedings in Informatics (LIPIcs), Volume 243, pp. 8:1-8:16, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022).* https://doi.org/10.4230/LIPIcs.CONCUR.2022.8

Khononov, V. (2025): Balancing Coupling in Software Design - Universal Design Principles for Architecting Modular Software Systems. *Addison-Wesley, Hoboken, NJ, USA*. ISBN 978-0-137-35348-4

Kopetz, H. (1998): The Time-Triggered Model of Computation. *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium. IEEE Computer Society, New York, N.Y., USA, pp. 168-177*. https://dl.acm.org/doi/10.5555/827270.829023

Kopetz H., Bauer G. (2003): The Time-Triggered Architecture. *Proceedings of the IEEE, Vol. 91, No. 1, January 2003*. DOI: 10.1109/JPROC.2002.805821. Downloadable from: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5e461d6b54890ae3bce70334393560aa9235a769

Kopetz, H., Ademaj, A., Grillinger, P., Steinhammer, K. (2005): The Time-Triggered Ethernet (TTE) Design. Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), Seattle, WA, USA. Downloadable from: http://www.ann.ece.ufl.edu/courses/eel6686_15spr/papers/TTE_Design.pdf

Kopetz, H. (2017): The Time-Triggered Model of Computation. *The Ptolemy Project, University of Berkeley, Berkeley, USA.* Downloadable from: https://ptolemy.berkeley.edu/projects/embedded/research/hsc/class.F03/ee249/publications/TTmodelofComp.pdf

Kopetz H., Steiner W. (2022): Real-Time Systems - Design Principles for Distributed Embedded Applications. 3rd edition. *Springer Nature Switzerland, Cham, Switzerland*. ISBN 978-3-031-11991-0. https://doi.org/10.1007/978-3-031-11992-7

Kröger, F., Merz, S. (2008): Temporal Logic and State Systems. *Springer-Verlag, Berlin, Germany*. ISBN 978-3-540-67401-6

Lee E. A., Woodcock J. (2023): Time-Sensitive Software. *Research Directions, Cyber-Physical Systems, Vol 1. Cambridge University Press, Cambridge, UK.* https://doi.org/10.1017/cbp.2023.1

Liu G. (2022): Petri Nets - Theoretical Models and Analysis Methods for Concurrent Systems. *Springer Nature Singapore, Singapore.* ISBN 978-981-19-6308-7. https://doi.org/10.1007/978-981-19-6309-4

Lokuciejewski P., Marwedel P. (2011): Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems. *Springer Science & Business Media B.V., Dordrecht, The Nederlands*. ISBN 978-90-481-9928-0. https://doi.org/10.1007/978-90-481-9929-7

Maier R., Bauer G., Stoger G., Poledna S. (2002): Time-triggered architecture -A consistent computing platform. *IEEE Micro, vol. 22, no. 4, pp. 36-45, July-Aug. 2002*. doi: 10.1109/MM.2002.1028474. Available at: https://ieeexplore.ieee.org/abstract/document/1028474

Martin, R.C. (2017): Clean Architecture - A Craftsman's Guide to Software Structure and Design. *Addison-Wesley, Boston, USA*. ISBN 978-0134494166

Maruf, A.A., Niu, L., Clark, A., Mertoguno, J.S., Poovendran, R. (2022): A Timing-Based Framework for Designing Resilient Cyber-Physical Systems under Safety Constraints. https://arxiv.org/abs/2208.14282

Milner, R. (1980): A Calculus of Communicating Systems. *Springer Verlag, Berlin, Germany. LNCS Vol 92.* Downloadable from: https://www.lfcs.inf.ed.ac.uk/reports/86/ECS-LFCS-86-7/ECS-LFCS-86-7.pdf

Moreno, R.P., Salcedo, J.L.V. (2006): Implementation of time Petri nets in real-time Java. *JTRES '06, Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems, pp. 178 – 187*. https://doi.org/10.1145/1167999.1168029

Muller R.A. (2016): Now - The Physics of Time. *W. W. Norton & Company, New York, NY, USA.* ISBN 978-0-393-28523-9

Murer S., Bonati B., Furrer, F.J. (2014): Managed Evolution - A Strategy for Very Large Information Systems. *Springer Verlag, Berlin, Germany*. ISBN 978-3-642-43131-9. https://doi.org/10.1007/978-3-642-01633-2

Nakagawa E.Y., Antonino, P.O. [Editors] (2024): Reference Architectures for Critical Domains - Industrial Uses and Impacts. *Springer Nature, Cham, Switzerland*. ISBN 978-3-031-16959-5. https://doi.org/10.1007/978-3-031-16957-1

Nicollin X., Sifakis J, (1991): An Overview and Synthesis on Timed Process Algebras. *Presented at CAV'91, Alborg Denmark, July 1991*. Downloadable from: https://www-verimag.imag.fr/PEOPLE/Joseph.Sifakis/overviewtimedprocalg-cav91.pdf

NIST (2017): Framework for Cyber-Physical Systems - Volume 3: Timing Annex. *NIST Special Publication 1500-203. National Institute of Standards and Technology, Gaithersburg, MD, USA.* https://doi.org/10.6028/NIST.SP.1500-203.

Núñez-Alvarez, J. R., Benítez-Pina, I., Acosta-Montoya, G., Pino-Escalona, A., Villafuela-Loperena, L. (2023): Design of an Integrated Automation & Control System Using Petri Nets - Case Study. *Journal of Applied Research and Technology, 21(2), 169–180.* https://doi.org/10.22201/icat.24486736e.2023.21.2.1562

Obermaisser, R. (Editor), 2011: Time-Triggered Communication. *CRC Press, Boca Raton, FL, USA*. ISBN 978-1-439-84661-2

Öztemür, S. (2015): Exceptions and Exception Handling in Business Process Management Systems - Analysis and Classification. *Ulm University, Ulm, Germany. Faculty of Engineering and Computer Science. Institute of Database and Information Systems.* Downloadable from: https://dbis.eprints.uni-ulm.de/id/eprint/1311/1/NA_Oez_2015.pdf

Pasandideh, S., Gomes, L., Maló, P. (2022): Modelling Cyber-Physical Social Systems Using Dynamic Time Petri Nets. *Faculty of Science and Technology, NOVA University of Lisbon, Centre of Technology and Systems - CTS, UNINOVA, Campus da Caparica, 2829-516 Monte Caparica, Portugal*. Downloadable from: https://research.unl.pt/ws/portalfiles/portal/12931496/Modelling_Cyber_Physical_Social_Systems_Using_Dynamic_Time.pdf

Penczek, W., Pólrola, A. (2006): Advances in Verification of Time Petri Nets and Timed Automata - A Temporal Logic Approach. *Springer Verlag, Berlin, Germany*. ISBN 978-3-540-32869-8

Philippou, A., Sokolsky, O., (2007): Process-Algebraic Analysis of Timing and Schedulability Properties. *Technical Paper*. Downloadable from: https://www.cs.ucy.ac.cy/~annap/papers/rtpa.pdf

Platzer, A. (2018): Logical Foundations of Cyber-Physical Systems. *Springer International Publishing, Cham, Switzerland.* ISBN 978-3-319-63587-3. https://doi.org/10.1007/978-3-319-63588-0

Popova-Zeugmann L. (2016): Time and Petri Nets. *Springer Verlag, Heidelberg, Germany*. ISBN 978-3-662-51435-1. https://doi.org/10.1007/978-3-642-41115-1

Power S.E. (2021): Philosophy of Time - A Contemporary Introduction. *Routledge Publishing, New York, NY, USA*. ISBN 978-1-138-24049-0

Rajeev, A.C., Mohalik, S., Ramesh, S. (2012): Verifying timing synchronization constraints in distributed embedded architectures. *2012 Design, Automation & Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 12-16 March 2012, pp. 200-205,* https://doi.org/10.1109/DATE.2012.6176463. Available at: https://ieeexplore.ieee.org/document/6176463

Richards, M., Ford, N. (2025): Fundamentals of Software Architecture - An Engineering Approach. *O'Reilly Media, Boston, USA*. ISBN 978-1-098-17551-1

Rodriguez, R.J., Julvez, J., Merseguer, J. (2013): On the Performance Estimation and Resource Optimisation in Process Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems. New York, USA. Vol. 43, No. 6, pp. 1385-1398, Nov. 2013.* https://doi.org/10.1109/TSMC.2013.2245118. Downloadable from: https://zaguan.unizar.es/record/57494/files/texto_completo.pdf

Rohr, M., (2015): Workload-sensitive Timing Behavior Analysis for Fault Localization in Software Systems. *PhD Thesis, Kiel University, Kiel, Germany. Printed by Books on Demand, Norderstedt, Germany*. ISBN 978-3-7347-4516-4

Rushby, J. (2002): An Overview of Formal Verification for the Time-Triggered Architecture. *Invited paper, presented at FTRTFT'02, Oldenburg, Germany, September 2002. Springer-Verlag, Heidelberg, Germany (LNCS Vol. 2469, pp. 83–105).* Downloadable from: https://depend.cs.uni-saarland.de/fileadmin/user_upload/depend/dnjansen/rushbyOverview.pdf

Rushby J. (2005): An Overview of The Time-Triggered Architecture (TTA) and its Formal Verification. *Computer Science Laboratory, SRI International, Menlo Park, California, USA*. Downloadable from: https://www.csl.sri.com/users/rushby/slides/kestrel05.pdf

Shaw, R., Jackman, B. (2008): An Introduction to FlexRay as an Industrial Network. *IEEE, New York, USA*. ISBN 978-1-4244-1666-0. Downloadable from: https://www.researchgate.net/publication/224349830_An_introduction_to_FlexRay_as_an_industrial_network

Shi, W., He, Z., Gu, C., Ran 3, N., Ma, Z. (2023): Performance Optimization for a Class of Petri Nets. *Sensors, Basel, Switzerland. 2023, January 28;23(3):1447*. https://doi.org/10.3390/s23031447

Shrivastava, A., Derler, P., Baboud, Y.S.L., Stanton, K., Khayatian, M., Andrade, H.A., Weiss, M., Eidson, J., Chandhoke, S. (2016): Time in cyber-physical systems. *CODES '16: Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Article No. 4, Pages 1 – 10. Pittsburgh, PA, USA*. https://doi.org/10.1145/2968456.2974012. Available from: https://ieeexplore.ieee.org/document/7750983

Sony, M. (2020): Design of cyber-physical system architecture for industry 4.0 through Lean Six Sigma - conceptual foundations and research issues. *Production and Manufacturing Research, Vol. 8, No. 1, 158–181*. https://doi.org/10.1080/21693277.2020.1774814. Downloadable from: https://www.tandfonline.com/doi/epdf/10.1080/21693277.2020.1774814?needAccess=true

Struthers R. (2024): Hands of Time - A Watchmaker's History of Time. *Harper-Collins Publishers, New York, NY, USA.* ISBN 978-1-529-33904-8

Thiele, L., Chakraborty, S., Naedele, M. (2000): Real-time calculus for scheduling hard real-time systems. *2000 IEEE International Symposium on Circuits and Systems (ISCAS), Geneva, Switzerland, 2000, pp. 101-104, vol.4*. https://doi.org/10.1109/ISCAS.2000.858698. Available from: https://ieeexplore.ieee.org/abstract/document/858698

Thramboulidis, K. (2010): IEC 61499 Function Block Model - Facts and Fallacies. *IEEE Industrial Electronics Magazine, New York, NY, USA, Vol. 3, Nr. 4, pp. 7 – 26.* https://doi.org/10.1109/MIE.2009.934788. Downloadable from: https://www.researchgate.net/publication/224089609_IEC_61499_Function_Block_Model_Facts_and_Fallacies

Ungureanu, G., Sander, I. (2017): A Layered Formal Framework for Modeling of Cyber-Physical Systems. *Design, Automation and Test in Europe Conference and Exhibition (DATE), Lausanne, Switzerland, 2017, pp. 1715-1720.* https://doi.org/10.23919/DATE.2017.7927270. Downloadable from: https://kth.diva-portal.org/smash/get/diva2:1114832/FULLTEXT02.pdf

Wang J. (1998): Timed Petri Nets - Theory and Application. *Springer Science+Business Media, New York, NY, USA*. ISBN 978-0-7923-8270-6. https://doi.org/10.1007/978-1-4615-5537-7.

Wang, Y. (2002): The Real-Time Process Algebra (RTPA). *Annals of Software Engineering 14, 235–274, 2002. Kluwer Academic Publishers, Amsterdam, The Netherlands*.

Willemse, T.A.C. (2003): Semantics and Verification in Process Algebras with Data and Timing. *IPA Dissertation Series 2003-05. Printed by University Press Facilities, Eindhoven, Netherlands*. ISBN 90-386-0672-9. Downloadable from: https://timw.win.tue.nl/articles/thesis.pdf

Wolf, F. (2002): Behavioral Intervals in Embedded Software - Timing and Power Analysis of Embedded Real-Time Software Processes. *Kluwer Academic Publishers*, *Boston*, *USA*. ISBN 978-1-402-07135-5.

Yoong, L.H., Roop, P.S., Salcic, Z. (2013): Implementing constrained cyber-physical systems with IEC 61499. *ACM Transactions on Embedded Computing Systems (TECS), Volume 11*, *Issue 4*, *Article No. 78, pp. 1 – 22*. https://doi.org/10.1145/2362336.2362345

Yoong, L.H., Roop, P.S., Bhatti, Z.E., Kuo, M.M.K. (2016): Model-Driven Design Using IEC 61499 - A Synchronous Approach for Embedded and Automation Systems. *Springer International Publishing, Cham, Switzerland. ISBN 978-3-319-10520-8*. https://doi.org/10.1007/978-3-319-10521-5