TU WIEN Informatics

Towards Dependable CPS/IoT Ecosystems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Haris Isakovic
Matrikelnummer 00325697

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Radu Grosu
Zweitbetreuung: Univ.-Prof. Ezio Bartocci

Diese Dissertation haben begutachtet:

Univ.Prof. Jerker Delsing	Univ.Prof. Manuel Wimmer

Wien, 3. August 2021

Haris Isakovic

TU WIEN Informatics

Towards Dependable CPS/IoT Ecosystems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Haris Isakovic
Registration Number 00325697

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Radu Grosu
Second advisor: Univ.-Prof. Ezio Bartocci

The dissertation has been reviewed by:

Univ.Prof. Jerker Delsing	Univ.Prof. Manuel Wimmer

Vienna, 3rd August, 2021

Haris Isakovic

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Haris Isakovic

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

Wien, 3. August 2021

Haris Isakovic

Acknowledgements

I would like to express my deepest gratitude to professor Radu Grosu for his support and mentorship during my PhD study. He inspired us to new heights and to reach the best in us both in our professional and personal lives. Further, I owe a big thank you to professors Ezio Bartocci and Peter Puschner for their cooperation and valuable advice. To my colleagues Christian, Armin, Oliver, Bernhard, Denise, and Ramin, thank you very much. You helped me numerous times, you have been there for me not only as colleagues but as dear friends. In the uncharted sea of projects I managed to find my way with selfless help of our secretaries Maria, Gerda and Viktoria.

I am thankful to all my friends and colleagues from projects EMC2, Productive4.0, CPS/IoT Ecosystem, ACROSS, MultiPartes, iDev4.0 and Adeptness. They shared with me their unreserved dedication, utmost professionalism, creative insights, unmatched expertise and those valuable moments of sincere friendship.

This thesis was a long process, and I am proud to say that I always had friends to rely upon. Through all my ups and downs, they never showed any doubt in my abilities and they gave me some of the best moments in my life.

I cannot express how fortunate I feel to have such a wonderful family behind me, from my sister Nerma, her husband Nedim, all aunts, uncles, cousins, to my late grandparents. I feel an incredible sense of respect for all they did for me on this journey, I hope I get the chance to return the favor soon.

My wife Azemina is the person who carried much of the burden with me, she is my fortress of solitude, my rock, and my safe harbor. This is the person who loved me from the first day we met, and never showed anything but relentless pursuit of support and respect. She pushed me when I felt giving up, and she lifted me up when I felt down. She gave me the biggest joy in my life, our daughter Nora. Together, they make every day in my life wonderful.

Two people without whom this would be unimaginable are my parents Enisa and Šefik. I dedicate this work to them because it is their unreserved sacrifice that allowed me to be here. They motivated and guided me my whole life, to reach this goal. Without a grain of doubt, thorough sickness and all the hardships, they kept giving even when their help was no longer needed. I will forever be grateful for all they did to make this possible.

Kurzfassung

Wir befinden uns am Anfang einer technologischen und sozioökonomischen Revolution, in der Computerintelligenz Teil gewöhnlicher physischer Objekte wird, soziale Interaktionen regelt und die menschliche Kognition im Allgemeinen beeinflusst. Wir betreten die Ära der Smart World! Intelligente Geräte werden zu integralen Bestandteilen industrieller Systeme, von Transport, Gesundheitswesen, Bauwesen, Chemie, Energie, Bildung und Unterhaltung. Wir stehen vor einer rasanten Zunahme vernetzter Rechenknoten im gesamten Spektrum industrieller Anwendungen und Allzweckanwendungen.

Der technologische Träger dieser Revolution ist die Synergie zwischen Cyber-Physischen Systemen (CPS) und dem Internet der Dinge (IoT). Diese beiden Konzepte spezifizieren Fähigkeiten von rechnergestützten Systemen, die physikalische Welt mithilfe von mathematischen Modellen zu interpretieren, die physikalische Umgebung zu beobachten und zu manipulieren sowie Informationen über verschiedene Einsatzbereiche hinweg zu kommunizieren.

In dieser Arbeit definieren wir das Konzept des CPS/IoT-Ökosystems als eine hierarchische Struktur, die Praktiken und Verfahren für die Modellierung, das Design, die Entwicklung, die Ausführung und den Betrieb von intelligenten Systemen regelt. Wir unterteilen diese Systeme in drei lose voneinander abhängige Betriebsbereiche: die Cloud, den Fog und den Swarm.

Weiter, wir schlagen eine Sammlung von Methoden und Ansätzen vor, die ein zuverlässiges Design, eine zuverlässige Ausführung und einen zuverlässigen Betrieb von CPS/IoT-Ökosystemen ermöglichen. Insbesondere beginnen wir mit Methoden zur Gewährleistung der deterministischen Ausführung von Aufgaben in sicherheitsrelevanten Anwendungen. Als Erweiterung dieser Notiz schlagen wir eine Virtualisierungstechnik für Kommunikationskanäle für Many-Core-Plattformen mit begrenzten physikalischen Schnittstellen und eine sichere Kommunikationsarchitektur vor, die auf integrierten Architektureigenschaften wie der zeitlichen und räumlichen Isolierung von Knoten beruht.

Das CPS/IoT-Ökosystem ist eine sehr heterogene Umgebung mit Hardware- und Softwarekomponenten, die von mehreren Organisationen entwickelt und implementiert werden. Um die Kohärenz zwischen verschiedenen Komponenten zu gewährleisten und die Komplexität zu reduzieren, schlagen wir ein kontinuierliches Integrations- und Bereitstellungsschema (CI/CD) für das CPS/IoT-Ökosystem vor. Darüber hinaus demonstrieren wir einen

Runtime Verification (RV) Mechanismus, der die Grundlage für die Orchestrierung der Dienstgüte (QoS) und die dynamische Rekonfiguration von CPS/IoT-Anwendungen bildet.

Am Ende schlagen wir Methoden vor, um ein energetisch nachhaltiges CPS/IoT-Ökosystem zu erreichen. Darüber hinaus haben wir ein Modell zur Vorhersage des Energieverbrauchs im gesamten CPS/IoT-Spektrum mit Rückblick auf die Smart-World-Revolution definiert. Um den durch smarte Geräte verursachten Energie-Overhead zu reduzieren, schlagen wir eine alternative Stromversorgungsmethode für CPS/IoT-Sensorknoten vor.

Abstract

We find ourselves at the beginning of a technological and socio-economic revolution, where computational intelligence is becoming part of ordinary physical objects, governing social interactions and affecting human cognition in general. We are entering the era of a Smart World! Smart devices are becoming integral parts of industrial systems, from transportation, healthcare, construction, chemical, energy, education and entertainment. We are facing a rapid increase of networked-computing nodes across the spectrum of industrial applications and of general-purpose applications.

The technological carrier of this revolution is the synergy between cyber-physical systems (CPS) and the Internet of Things (IoT). They specify the abilities that computational systems need to possess, in order to interpret their physical world by using mathematical models, to observe and manipulate their physical environment through sensors and actuators, and to communicate information across different scopes of operation.

In this thesis we define the concept of CPS/IoT Ecosystem as a hierarchical structure, that governs practices and procedures for modeling, design, development, execution and operation of smart systems. We divide these systems in three loosely dependent scopes of operation: the cloud, the fog, and the swarm.

Furthermore, we propose a series of methods and approaches that support the dependable design, execution, and operation of CPS/IoT Ecosystems. In particular, we start with methods ensuring the deterministic execution of tasks in safety constrained applications. As an extension to these methods we propose a virtualization technique for communication channels for many-core platforms with limited physical interfaces, and a secure communication architecture that relies on integrated architectural properties such as temporal and spatial isolation of nodes.

A CPS/IoT Ecosystem is a highly heterogeneous environment with hardware and software components that are designed and implemented by multiple organizations. To ensure coherence between different components and to reduce complexity we propose a continuous integration and deployment (CI/CD) scheme for CPS/IoT Ecosystem. Furthermore, we demonstrate a runtime verification (RV) mechanism that provides a basis for quality of service (QoS) orchestration and dynamic reconfiguration of CPS/IoT applications.

As final step in this thesis we propose methods to achieve energy-sustainable CPS/IoT Ecosystems. Moreover, we defined a model to predict energy consumption across the

CPS/IoT spectrum with retrospect to the Smart-World revolution. To reduce energy overhead created by smart devices we propose an alternative energy consumption method for CPS/IoT sensor nodes.

In conclusion, this thesis tries to seed methodological guidelines on how to build dependable CPS/IoT Ecosystems for applications with various confidence requirements. We want to understand the upcoming changes and reduce eventual effects of ad-hoc development. To explain physical environments using mathematical models and to learn new emerging behaviors using this massive incursion of new data and new insights.

Contents

Preface

The purpose of this cumulative doctoral thesis is to motivate and summarize the research work I have undertaken as PhD student at the CPS division of the faculty of informatics, TU Wien. During this time, my work took place within several CPS/IoT research projects, and it resulted in thirteen publications.

This thesis unifies all the individual research activities as a concerted work, that explores how to build dependable systems within a CPS/IoT Ecosystem, and which shows how to achieve this goal by using a structural hierarchy of computational systems. In short, the individual topics explored in our published work, advance the design, development, and operation of dependable CPS/IoT Ecosystems.

The thesis is constructed from two parts: 1) An introductory chapter that explains the motivation and methodology behind the topic of dependable CPS/IoT Ecosystems, and, 2) The relevant publications sorted as chapters in the thesis. Individual works were originally motivated and performed by different ideas and requirements. However, with this work we want to highlight how they fit in a larger picture and how they contribute to the holistic concept of a dependable CPS/IoT Ecosystem.

The diversity of topics, methods, and tools involved in this work is compelling. We want both to introduce the readers to the important aspects of each publication, and to establish an understanding of how these concepts contribute to the idea of a dependable CPS/IoT Ecosystem. For this purpose, we devised a side-note method, which is created from three types of notes, denoted by the questions Why?, What? and How?

Why? This note describes a motivational aspect of a subject, and it reminds the reader of the importance behind the statement, paragraph, or section towards greater good.

What? Each of these notes identifies a requirement to be fulfilled by the CPS/IoT Ecosystem or a challenge to be solved in order to establish a premise, or answer a question on the necessary steps towards a solution.

How? These notes will identify concrete solutions for the problems and remind users how these solutions correspond to given research questions.

CHAPTER 1

Introduction

1.1 Motivation

Advances in science and technology in the last few decades have resulted in tremendously complex systems that extend over different disciplines, physical and temporal scales, and involve various individuals or organizations.

This thesis explores how cyber-physical systems are entangled within a complex web of the physical environment, the Internet, and the ability to maintain specific safety and security criteria. We define a CPS/IoT Ecosystem as a heterogeneous structure of hardware devices and corresponding software components distributed over three intertwined scopes of operation: the swarm, the fog, and the cloud (see Figure 1.1) [66].

CPS/IoT Ecosystems can be seen as a result of natural evolution, similar to the one that happened for biological systems, where single cells merged over time into tissues/organs/bodies to perform more complex tasks. The swarm of sensors/actuators, akin to the human skin/muscle cells, produces the big (real-time) data through its sensors, and consumes it with its actuators. This data is shared with the fog. The fog, akin to the spinal cord, controls the sensors and the actuators in real time. It is capable of doing mid-scale computational tasks and storing intermediate and temporary data for purposes of aggregation or filtering. The cloud, akin to the human brain, possesses theoretically unlimited storage and computation resources. It is here where the data is stored and where planning and other offline tasks such as machine learning mainly happen.

By combining these individual scopes of operation we can increase the capability of the system by a large margin, yielding new emerging functions and enabling new use cases. These new features create new opportunities but also increase overall complexity. A perfect example for this can be found in the automotive industry. The complexity of automotive software has increased exponentially over the past 40 years (see Figure 1.2). As the systems get more complex and more conservative in terms of dependability,

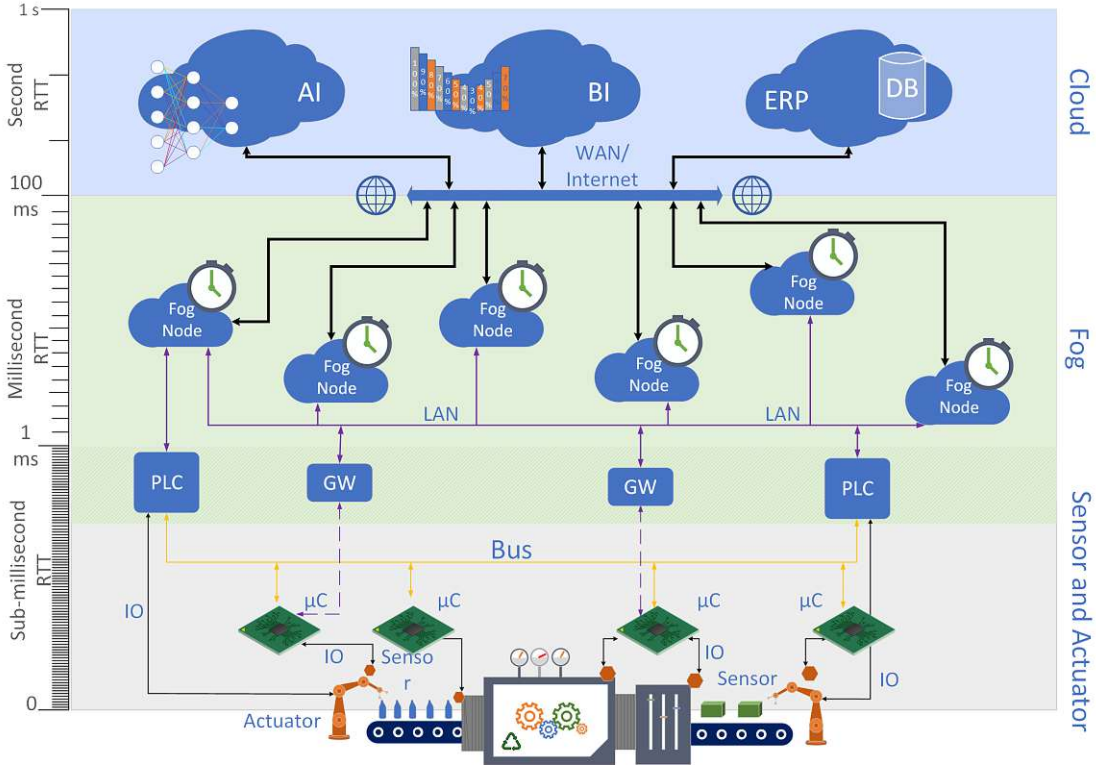


Figure 1.1: The architecture of a CPS/IoT Ecosystem. (Right) Three scopes of operation Cloud, Fog (Edge), Swarm (Sensors/Actuators). (Left) Time scale for the message exchange and reaction. Abbreviations: Artificial Intelligence (AI). Business Intelligence (BI). Enterprise Resource Planning (ERP). Data-Base (DB). Wide Area Network (WAN). Local Area Network (LAN). Programmable Logic Array (PLC). Communication Gateway (GW). Micro-controller (μC). Input-Output (IO). Round-Trip Time (RTT).

these tasks are getting more difficult. The higher-dependability standards are essential as we wander from general-purpose applications towards industrial safety-constrained applications, such as car-2-infrastructure or industrial control.

Why are these new technologies necessary in already functional systems? We can look at this from two perspectives: Functional and Economical. In automotive industry, for example, these new features significantly improve the cars in terms of safety, convenience, and comfort. Generally, the functional motivation for this revolution can be summarized in the following statements: a) Increase general functionality and capabilities of a system, b) Increase system efficiency, c) Achieve emergent insights into a system.

If we observe this change from an economic perspective, this revolution creates brand-new markets, innovation in business models, reduces efforts and overheads, and enables a tighter bond between management and the field level of operation. For example, using a continuous integration and delivery methods can save up to 78% of development costs

[58]. By applying advanced machine learning methods on sensor data collected from the physical environment of a machine it is possible to improve failure detection of robust mechanical systems by a significant margin, thus reducing maintenance and eventual downtime costs [57].

To conclude, the rise in complexity is inevitably advancing, and it is arguably justified. This raises the question: "How can we maintain the dependability of these systems despite the increase in complexity?" This thesis proposes a series of technologies that support the development of some of the missing pieces in the CPS/IoT puzzle.

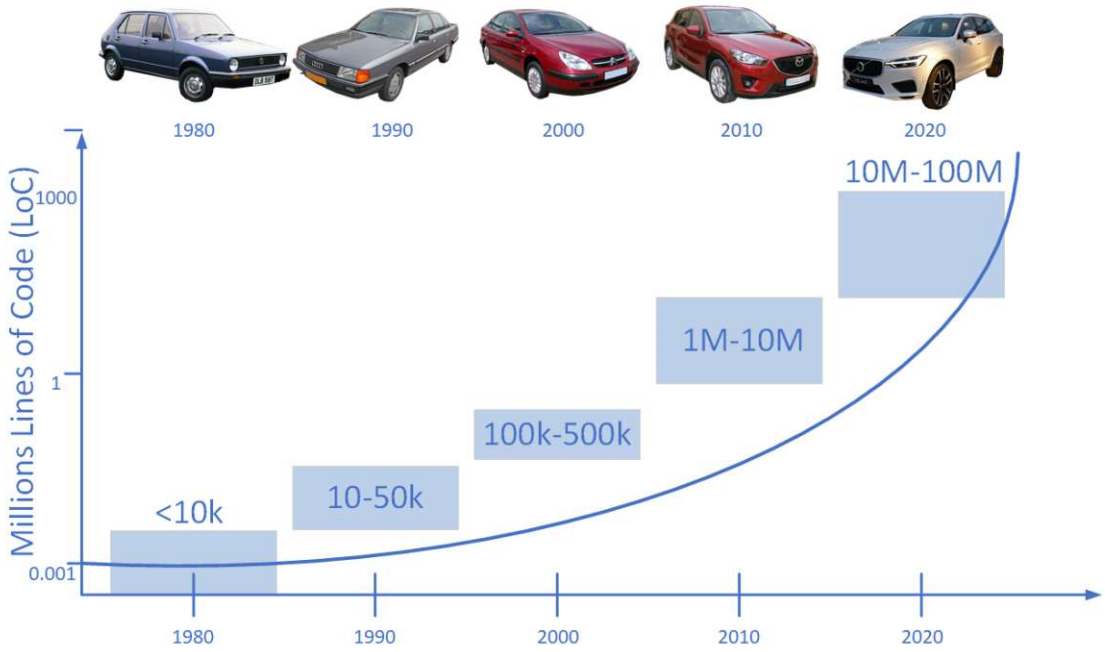


Figure 1.2: Software complexity in automotive sector, in terms of lines of code (LoC).

Why? A Smart World will require an enormous number of devices over next decade. An ad-hoc implementation of the smart world could potentially lead to chaos. It could result in an incomprehensible complexity, one which could endanger safety-critical industrial systems, economic institutions, and personal privacy.

1.2 Problem Statement and Research Gap

The three scopes of operation within a CPS/IoT Ecosystem, are fully capable of operating individually. In fact they have been observed and explored as separate entities in the past. However, with an increasing ability to analyze and understand our physical environment, new methods for developing efficient CPS are emerging. A necessary requirement for this new paradigm is constant communication, both horizontally and vertically. Horizontal

communication happens within a scope of operation, among components or devices, through a bus or LAN. As a vertical hierarchy, a CPS/IoT Ecosystem establishes a functionally relevant vertical communication, from the tiniest device to the Cloud.

The design and development process for each scope of operation is different from the others; they are loosely integrated, with numerous gaps in-between. This work explores the ability of these systems to facilitate dependability, both as an assembly of its defining properties, and also through a perspective of individual properties.

The dependability of computing systems was initially defined either as *"the ability to deliver service that can justifiably be trusted"* or as "*the ability to avoid service failures that are more frequent and more severe than is acceptable"* [28]. It is commonly accepted that the dependability of a computing system is determined by six attributes: availability, reliability, safety, integrity, and maintainability [28]. The availability attribute can be described as a readiness of the system to deliver correct service. The reliability attribute is a probabilistic measure of the system's ability to deliver correct service in a certain period of time. The safety attribute is augmenting reliability in terms of avoiding catastrophic failures that could result in a loss of a life, environmental or economic damages. Integrity is an attribute that guarantees that a system is not improperly or maliciously altered. Finally, the ability of a system to be properly repaired or modified in its life-cycle is referred to as maintainability.

More recent definitions of dependability include security as an attribute that comprises integrity and confidentiality [70][28]. This means that dependability and security largely overlap, and in the extension of the text we will assume dependability encompasses security.

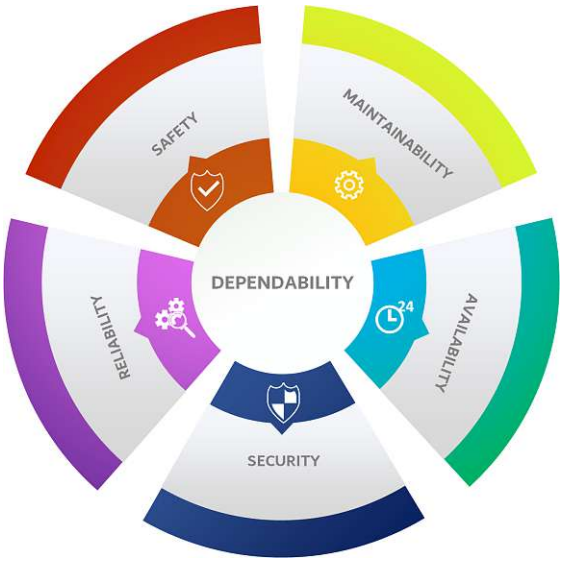


Figure 1.3: Dependability attributes.

Examples for dependable systems are traditionally drawn from industries with a strong safety, economic or environmental impact in case of catastrophic failure, such as aerospace, avionics, railway or automotive [49]. However, in recent years the scopes of operations of these systems have expanded into other domains, such as updates over the internet, multimedia, and communication with civil infrastructure.

Automotive vehicles are increasingly linked with cloud servers over the Internet. They perform tasks such as monitoring, multimedia streaming, and even over the air updates [78]. Figure 1.4 shows the gradual evolution of on-board architectures in the automotive sector, and the importance of connectivity with the Internet and infrastructure. Another

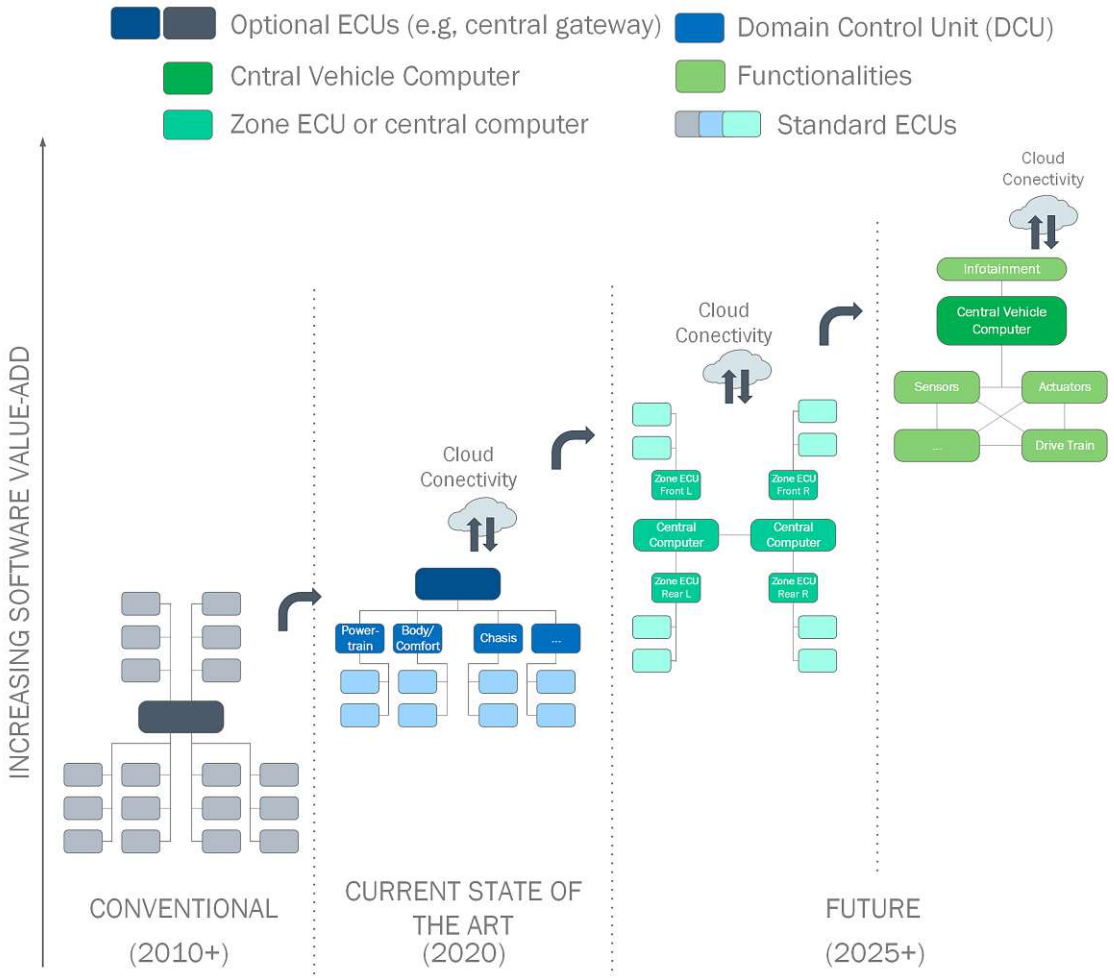


Figure 1.4: Evolution of on-board architectures in the automotive sector [44].

example is the industrial Internet-of-Things (IoT), where entire industrial processes are interlinked over multiple scopes of operation [46].

In hindsight, the dependability definition can be further extended to include attributes such as performability [84], sustainability [39], testability [84], explainability [40], scalability [72] and reconfigurability [86]. One common conclusion is that these attributes address issues that emerge as a result of overuse or extended usage time, interconnection with other systems, or application in environmentally sensitive settings. The dependability of computing systems needs to be adapting to emerging behaviors and system structures (e.g., IoT, quantum computing). With this in mind, we would like to revisit dependability considerations, standardization, and best practices in the domain of CPS/IoT Ecosystem.

As defined in Section 1.1 of the document we assume that a CPS/IoT Ecosystem (see Figure 1.1) can be divided in three hierarchical scopes of operation: the cloud, the fog,

and the swarm. Each individual system within this hierarchy is defined by a set of different structural, resource, and functional properties. Their purpose and application will determine the required level of dependability or standard conformity that needs to be achieved. For example, the same functionality can be achieved with multiple devices, but only certain devices are certified for a certain level of dependability or security.

Further, a CPS/IoT Ecosystem is determined by its intended function, and that is to establish a reliable connection to the physical environment, collect data, analyze it, and learn emerging behaviors that can be used to increase efficiency or ensure entirely new functions of a certain system. On the other hand, this increases the overall risk of emerging faults, errors, and failures.

The level of co-dependency of individual components yields new emerging behaviors that are often not considered in the design of the individual components and their failure model. Thus, when we explore dependability we need to consider how actions of individual components resonate on the global scale of a CPS/IoT Ecosystem. Linking systems such as vehicles to the Internet increases security risks that can have a significant impact on the safety and dependability of the system in general [37].

In conclusion, we would like to highlight challenges that are developing with the evolution of computational systems in terms of dependability:

- Categorize dependability requirements of CPS/IoTs, as we need to adapt standards and best practice methods to include emerging new risks.
- Ensure dependability not only within the scope of an individual component, but also within the scope of the entire system.
- Re-evaluate the dependability model in use, and explore new properties, for example sustainability, which are becoming more important with the increase in connectivity and the quantity of computational units.

We will expand on these topics in Section 1.3 as we define specific research questions.

Why? In this section we argued for the necessity of dependable COS/IoT Ecosystems through examples and hinted to the gaps in the research on related topics.

1.3 Aim of the Thesis

This thesis proposes a set of research questions highlighting CPS/IoT properties required for dependable operation. We emphasize individual research questions that identify roadblocks on this path, in addition to the state-of-the-art technologies that can be used to mitigate these challenges, and create emerging concepts. Together, they create a baseline for dependable design, development, and operation of CPS/IoT applications.

RQ1: How to build dependable hardware architectures for CPS/IoTs?

Table 1.1: Mapping research questions to dependability attributes.

Attributes	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6	RQ7
Testability	✓		✓				
Sustainabilty	✓		✓		✓		✓
Performability		✓				✓	
Explainability	✓		✓		✓	✓	✓
Scalability		✓	✓		✓		
Reconfigurability					✓	✓	
Availability	✓	✓	✓			✓	✓
Reliability	✓		✓	✓		✓	
Safety	✓		✓	✓		✓	
Confidentiality				✓		✓	
Integrity				✓	✓	✓	
Maintainability					✓	✓	✓

Multi-core commercial off-the-shelf (COTS) processors are designed for optimal performance in systems with no critical consequences (s.a., death, economic loss) in case of failure. Primitive non-deterministic scheduling and other similar features make the multi-core architectures not viable for safety-critical applications, if used in high-performance multi-core modes of operation. Ensuring dependable execution while maintaining acceptable performance capabilities is a rather difficult challenge.

RQ2: How to achieve virtualization of hardware resources in time?

A deterministic multi-core processor architecture has limited interface access due to safety and security requirements. Thus, individual cores may run into race conditions for hardware resources if not clearly divided between individual cores.

RQ3: How to enable dependable mixed-criticality integration in many-core architectures?

Deterministic multi-core systems provide excellent conditions for safety constrained systems. However, state-of-the-art safety constrained systems are increasingly more entangled with non-critical systems. The non-critical systems commonly require higher performance factors, and they communicate with safety components.

RQ4: How to achieve a transparent and secure communication architecture on integrated MPSoC?

Deterministic MPSoCs are built upon the notion of strictly isolated components, including processing and network components. This baseline can be used to achieve secure communication both on and off the chip with little to no overhead on application nodes. We propose a secure channel architecture where secure information is completely hidden from the application cores, and all security services are offered by a trusted secure core.

RQ5: How to conceptualize synergies between CPS and IoT?

CPS is a computational system that interacts closely with the physical world either through perception or manipulation. A CPS is commonly represented by a control loop monitoring a physical system; based on the model of that system, it creates decisions that control it. IoT is a networking concept that creates a link between CPSs independent of its size or function. These systems are composed together in system-of-systems overlapping computing and communication infrastructure. This results in a highly-complex structure of heterogeneous devices and software components, that is progressively harder to interpret, grasp and analyze. The question that arises is how can we hierarchically structure CPS/IoTs in a way we can understand this composition to the point that it can be used in applications requiring a high level of confidence?

RQ6: How to ensure QoS through run-time verification and monitoring on the entire scale of operation of a CPS/IoT Ecosystem?

As the system gets more complex we need to apply automation and reduce manual effort in development and maintenance, in order to avoid faults caused by human error or process desynchronization. This would increase the overall ability to keep a system stable for a longer periods, and significantly reduce downtime. How can we increase QoS by reducing manual effort in development and using exiting infrastructure supporting run-time verification methods onto these processes?

RQ7: How to achieve sustainable CPS/IoT Ecosystems?

The energy component is a significant cost overhead in industrial production. The introduction of swarm-like systems with billions of devices makes this cost overhead to an important and ubiquitous problem for the industry. Reducing the energy footprint of computational devices would increase their overall dependability as they would ultimately be less of a liability economically and environmentally speaking.

1.4 Publications

In order to give an overview of the publications relevant to this thesis we list them and provide a short name that will be used in the remainder of this document as a reference. We also provide a short discussion about the author's contributions to the individual papers. Publications marked with * are included in the thesis.

I **(ACROSS MPSoC)** The ACROSS MPSoC - A New Generation of Multi-Core Processors designed for Safety-Critical Embedded Systems / C. El Salloum, M. Elshuber, O. Höftberger, H. Isakovic, A. Wasicek, Talk: DSD 2012 (Euromicro Conference on Digital System Design), Cesme, Izmir, Turkey (invited); 2012-09-05 - 2012-09-08; in: "2012 15th Euromicro Conference on Digital System Design (DSD 2012), Proceedings", IEEE Computer Society, (2012), ISBN: 978-1-4673-2498-4; 105 - 113 [88].

II **(ACROSS MPSoC (Jurnal))** The ACROSS MPSoC - A new generation of multi-core processors designed for safety-critical embedded systems , C. El Sal-

loum, M. Elshuber, O. Höftberger, H. Isakovic, A. Wasicek, Microprocessors and Microsystems, 37 (2013), 8, Part C; 1020 - 1032 [47].

III-* **(Secure Channels)** Secure Channels in an Integrated MPSoC Architecture / H. Isakovic, A. Wasicek / Talk: 39th Annual Conference of the IEEE Industrial Electronics Society, Wien; 2013-11-10 - 2013-11-13; in: "Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE", (2013), ISSN: 1553-572x; 4488 - 4493 [67].

IV-* **(Virtual CAN)** Virtual CAN Lines in an Integrated MPSoC Architecture, A. Wasicek, O. Höftberger, M. Elshuber, H. Isakovic, A Fleck, Talk: 17th IEEE International Symposium on Object/Component-Oriented Real-Time Distributed Computing (ISORC), Reno, Nevada, USA; 2014-06-08 - 2014-06-12; in: "Proc. of the 12th IEEE International Conference on Industrial Informatics", (2014), ISSN: 1555-0885; 158 - 165 [93].

V-* **(Hybrid MPSoC)** A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform / H. Isakovic, R. Grosu, Talk: 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, USA; 2016-06-08 - 2016-06-10; in: "IEEE 25th International Symposium on Industrial Electronics (ISIE)", IEEE, (2016), ISSN: 2163-5145; 244 - 253 [63].

VI **(EMC^2 Project)** The EMC2 Project on Embedded Microcontrollers: Technical Progress after Two Years / W. Weber, A. Hoess, J. van Deventer, F. Oppenheimer,R. Ernst, A. Kostrzewa, P. Doré, T. Goubier, H. Isakovic, N. Druml,E. Wuchner, D. Schneider, E. Schoitsch, E. Armengaud, T. Söderqvist, M. Traversone, S. Uhrig, J.C. Pérez-Cortés, S. Saez, J. Kuusela, M. van Helvoort, X. Cai, B. Nordmoen,G.Y. Paulsen, H.P. Dahle, M. Geissel, J. Salecker, P. Tummeltshammer. 2016 Euromicro Conference on Digital System Design (DSD), 2016, pp. 524-531, doi: 10.1109/DSD.2016.72. [94].

VII-* **(Mixed-Criticality)** A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform / H. Isakovic, R. Grosu / in: "Solutions for Cyber-Physical Systems Ubiquity", 1; issued by: IGI Global; IGI Global, Hershey PA, USA 17033, 2018, (invited), ISBN: 9781522528456, 169 - 194 [64]

VIII **(HW for Safety)** A Survey of Hardware Technologies for Mixed-Critical Integration Explored in the Project EMC2 / H. Isakovic, R. Grosu, D. Ratasich, J. Kadlec, Z. Pohl, S. Kerrison, K. Georgiou, N. Druml, L. Tadros, F. Christiansen, E. Wheatley, B. Farkas, R. Meyer, M. Berekovic / Talk: SAFECOMP 2017 DECSoS, Trento (invited); 2017-09-12 - 2017-09-15; in: "Computer Safety, Reliability, and Security", Lecture Notes in Computer Science / Springer, Volume 10486 (2017), ISBN: 978-3-319-66284-8; 124 - 140 [65].

IX **(Self-Healing)** A Self-Healing Framework for Building Resilient Cyber-Physical Systems / D. Ratasich, O. Höftberger, H. Isakovic, M. Shafique, R. Grosu / Talk: 20th IEEE International Symposium on Real-Time Computing (ISORC 2017), Toronto, Canada; 2017-05-16 - 2017-05-18; in: "Real-Time Distributed Computing (ISORC), 2017 IEEE 20th International Symposium on", IEEE, (2017), ISBN: 978-1-5386-1574-4; 133 - 140 [86].

X-* **(CPS/IoT Ecosystem)** CPS/IoT Ecosystem: A Platform for Research and Education / H. Isakovic, D. Ratasich, C. Hirsch, M. Platzer, B. Wally, T. Rausch, D. Nickovic, W. Krenn, G. Kappel, S. Dustdar, R. Grosu / Talk: 14th Workshop on Embedded and Cyber-Physical Systems Education (WESE 2018), Turin, Italien; 2018-10-04 - 2018-10-05; in: "Cyber Physical Systems. Model-Based Design. 8th International Workshop on Model-Based Design of Cyber Physical Systems (CyPhy 2018), and 14th International Workshop on Embedded and Cyber-Physical Systems Education (WESE 2018)", R. Chamberlain, W. Taha, M. Törngren (ed.); Springer International Publishing, (2019), ISBN: 978-3-030-23702-8 [66].

XI **(Simulation for IoT)** Sensyml: Simulation Environment for large-scale IoT Applications / H. Isakovic, R. Grosu, B. Wally, T. Rausch, S. Dustdar, G. Kappel, D. Ratasich, V. Bisanovic / Talk: 45th Annual Conference of the IEEE Industrial Electronics Society (IECON 2019), Lisbon, Portugal, Portugal; 2019-10-14 - 2019-10-18; in: "IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society", IEEE Xplore, (2019), ISBN: 978-1-7281-4878-6; 3024 - 3030 [54].

XII **(Smart Farming)** CPS/IoT Ecosystem: Indoor Vertical Farming System, H. Isakovic, R. Grosu, A. Fasching, L. Punzenberger / Talk: 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT), Ancona, Italien; 2019-06-19 - 2019-06-21; in: "2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)", IEEE Xplore, (2019), ISSN: 0747-668x; 47 - 52 [55].

XIII-* **(QoS for IoT)** QoS for Dynamic Deployment of IoT Services, H.Isakovic, L.L. Ferreira, Irmin Okic, A. Dukkon, and R. Grosu, IEEE 22. International Conference on Industrial Technology ICIT 2021, Valencia, Spain [56].

XIV-* **(Sustainable IoT)** An Energy Sustainable CPS/IoT Ecosystem, H. Isakovic, E.A. Crespo and R. Grosu,(2021), EAI Edge-IoT 2020, In Paiva S., Lopes S.I., Zitouni R., Gupta N., Lopes S.F., Yonezawa T. (eds) Science and Technologies for Smart Cities. SmartCity360° 2020. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. [61].

XV **(Adaptive Filters)** Adaptive Signal Filtering Platform for a CPS/IoT Ecosystem, H. Isakovic, S. Dangl, Z. Tucakovic, R. Grosu, Special Session on the IEEE 22. International Conference on Industrial Technology ICIT 2021, Valencia, Spain [62].

1.5 Methodology

In the scope of this thesis we will elaborate the main methods for achieving dependable CPS/IoT Ecosystems, provide an extensive overview of the state-of-the-art of scientific and industrial literature, and highlight the importance of our research and relevant publications towards this goal.

The previous sections introduced CPS/IoT Ecosystems as complex hierarchical structures establishing basic causal relations among its structural subunits. Individual components in CPS/IoT Ecosystems are designed and implemented as independent units. Their functions have causal effects on the other components and their basic properties. The notion of CPS/IoT Ecosystem follows the definition of CPSoS as mentioned earlier; however, it expands the typical CPSoS definition further from the perspective of uncertainty and entanglement between components with higher requirements on functional safety and components with low or no-requirements for functional safety.

1.5.1 General Requirements

In order to understand the requirements behind dependable CPS/IoT Ecosystems, it is necessary to explain the recent history in the development of cyber-physical systems applied in manufacturing, automotive and other industrial sectors with strict functional rules and regulations. Initiatives such as Industry4.0 [8] from the German public and industrial authorities proposed radical revolution of industrial manufacturing systems towards functionally smarter and more efficient systems.

This revolution of industrial systems is supposed to create new values, business models, and improve overall social and economic benefits [69]. Another example where the industrial evolution is pushing the boundaries of technological and scientific development of cyber-physical systems is coming from the automotive industry. In particular, the introduction of autonomous and smart driving creates major challenges and yields extensive changes in the way how cars and their computer systems are implemented.

These new applications require that we transfer progressively more safety functions to fully automated systems (e.g., lane keeping, adaptive braking). They also require us to continuously increase the number of interconnections between local components and the outside world. For example, in the factory we are connecting in-factory systems with external software such as resource planning, accounting, and data analysis. The vehicles are connected to the surrounding infrastructure, or they are using the connection to the internet or smartphones to implement a smart in-vehicle instrumentation.

In order to achieve all these new features, systems need to be augmented by increasing connectivity, interoperability, and ubiquity while maintaining functional safety, security, and resource efficiency. To achieve the goals of Industry40, smart industrial systems must not only reach new levels of functional intelligence, but also improve on standardization and create new reference architectures, complexity management, and integration with other fields of operation such as supply chains, work organization and training, and regu-

latory supervision. Such systems must also increase overall connectivity by establishing broadband infrastructure for industry [69].

The evolution of cyber-physical systems and their ambient applications yields new technologies, methods and tools, under standardized guidelines and regulations. These systems were defined by properties and requirements proposed in the traditional dependability model [28]. However, new features and functions are producing emerging properties that are giving us a more enhanced description of the system.

What? The society constantly imposes new requirements on industrial systems; these trends are also reflected in computer systems. The computer systems need to be designed in a way that allows them to be easily adapted or upgraded to meet emerging needs of industry and society.

Architectural Significant Requirements (ASR)

The success of a system is pre-conditioned with the requirements that describe its function, behavior, and relations to other systems. The requirements that determine and shape the architecture of the systems can be defined as architectural significant requirements (ASR) [41]. In Section 1.2 we introduced a standard dependability model extended with additional attributes that emerged over time and that influence the systems ability to ensure a dependable operation.

In the scope of our work we observed emerging system properties that are going beyond the traditional dependability model and are vital for a dependable CPS/IoT Ecosystem:

- **Mixed-criticallity** the ability of the system to combine functionalities with high importance ("critical") for the correct operation of the system with others that might not be directly connected to the system's safe operation [33]. For example, to allow smart in-vehicle instrumentation that can manage both safety-critical functions and non-critical functions it is necessary to enable novel hardware architectures capable of separating these functions in space and time, in order to reduce possible interference to an absolute minimum [47].
- **Virtualization** is another property that is not included in the traditional model of computer-system dependability [28]. However, it is essential for most state-of-the-art computer systems. It is the function that allows a system *"to divide resources on multiple execution environments, using one or more concepts or technologies"* [81][91]. Use of virtualization as a means for achieving spatial isolation between components is an essential requirement in safety- and security-constrained systems.
- **Quality-of-Service (QoS)** stems from the communication aspect of computer systems, and it originally referred to the ability of the network to provide a service under specified requirements [42]. It is regulating a collection of properties such as throughput, latency, jitter, etc. The concept extended further in application design, and it became exclusively prominent in Internet applications, such as multimedia

streaming [25]. Industrial applications and the applications with safety constraints such as automotive, are strictly defined by their specifications. The satisfaction of these conditions is determining their QoS level. This is a collection of requirements from communication, hardware behaviour, energy consumption, software execution matrices, and resource management. A QoS management component or service is an essential part of any state-of-the-art industrial software framework [26].

- **Sustainability** is a property that surpasses its definition as a technical requirement. Moreover, it can be observed as a global socio-economic challenge [36]. We can observe sustainability from multiple aspects, such as: a) technical or longevity of the system; b) environmental or impact of on a surrounding ecosystem; c) economic or ability to maintain positive financial balance; d) individual or the ability of an entity to progress and develop in an environment; e) and social that propagates communication between individuals and organizations and their ability to resolve conflicts potential disputes [35]. In the scope of CPS/IoT Ecosystem it is necessary to reflect on two requirements, reducing power consumption and increasing the operational lifespan of each device, software or system in general.

- **Security** is not a novel or emerging property in terms of definition and function. It is included in traditional dependability [28]. Moreover, it is one of the properties with a largest set of requirements and highest rate of change. In the scope of a CPS/IoT Ecosystem where we have a high number of heterogeneous devices, software components and communication protocols it is extremely difficult to envision a unified solution satisfying all security requirements [59]. There are two main aspects when observing security in CPS/IoT Ecosystem. The first is local, where we need to establish cyber-resilience of each individual component [77]. The second aspect is global, where we need to ensure security frameworks for secure design and implementation of CPS/IoTs [29]. In conclusion we state that a secure design process starts with a secure design of hardware devices and ends with the security-proofing the application, both from cyber and physical aspect.

What? We identified the above-defined properties as essential to achieve certain aspects of dependability in a CPS/IoT Ecosystem.

Quality Assurance and Standardization

Standards are means of regulation compiled in a document and approved by authorized experts as a rule or guideline to establish a certain level of quality, order, or functionality for an entity or organization in a given context [27]. Standards provide three types of information: a) Normative, or prescriptive measures one needs to take to conform to a standard, b) Informative, or descriptive knowledge that supports conceptual understanding of the topic, and c) Requirement, which are the criteria needed to satisfactorily comply with a standard [6].

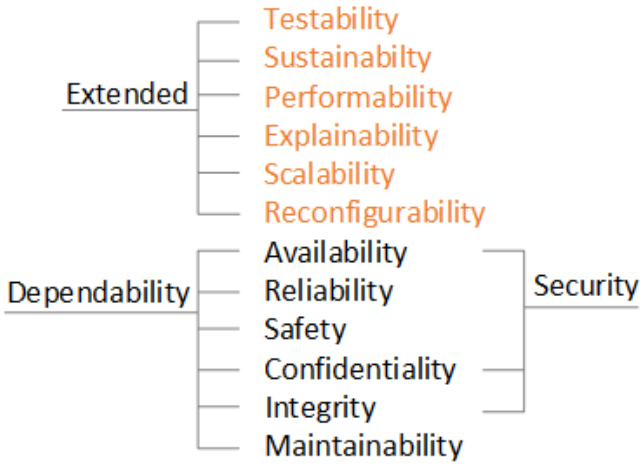


Figure 1.5: The extended model for dependability with emerging properties.

Standards are characterized depending on the domain of application, intended use, and geographical and social circumstances. International standardization bodies that are relevant to CPS/IoT Ecosystems are:

- International Electrotechnical Commission (IEC) [9],
- International Organization for Standardization (ISO) [10],
- International Telecommunication Union (ITU) [11],
- Institute of Electrical and Electronics Engineers (IEEE) [19],
- Association Connecting Electronics Industries (IPC) [23],
- International Society of Automotive Engineers (SAE) [17], and
- World Wide Web Consortium [21].

These organizations provide regulations and guidelines for electronic systems, telecommunication and computer systems on a global scale. Different regions or countries will have their respective standardization bodies e.g., National Institute of Standards and Technology (NIST) [13] or European Telecommunications Standards Institute (ETSI) [6] where standards can somewhat differ from international norms.

Standards relevant to CPS/IoT Ecosystems are mainly inherited from other domains. A major component are systems for embedded systems and development. Further, they are strongly influenced by the application domain where they are deployed, especially regarding the safety aspect. Table 1.2 shows a collection of standards that in one way or another support dependable design and implementation of CPS/IoT.

There is an ongoing initiative to standardize IoT in particular, with three standards that would regulate dependability norms, guidelines, and requirements for IoT systems. The standard marked as IEEE P2413 [7] is dealing with architectural frameworks for IoT. Further, the standard IEEE P1451 [14] works on the harmonization of IoT devices and

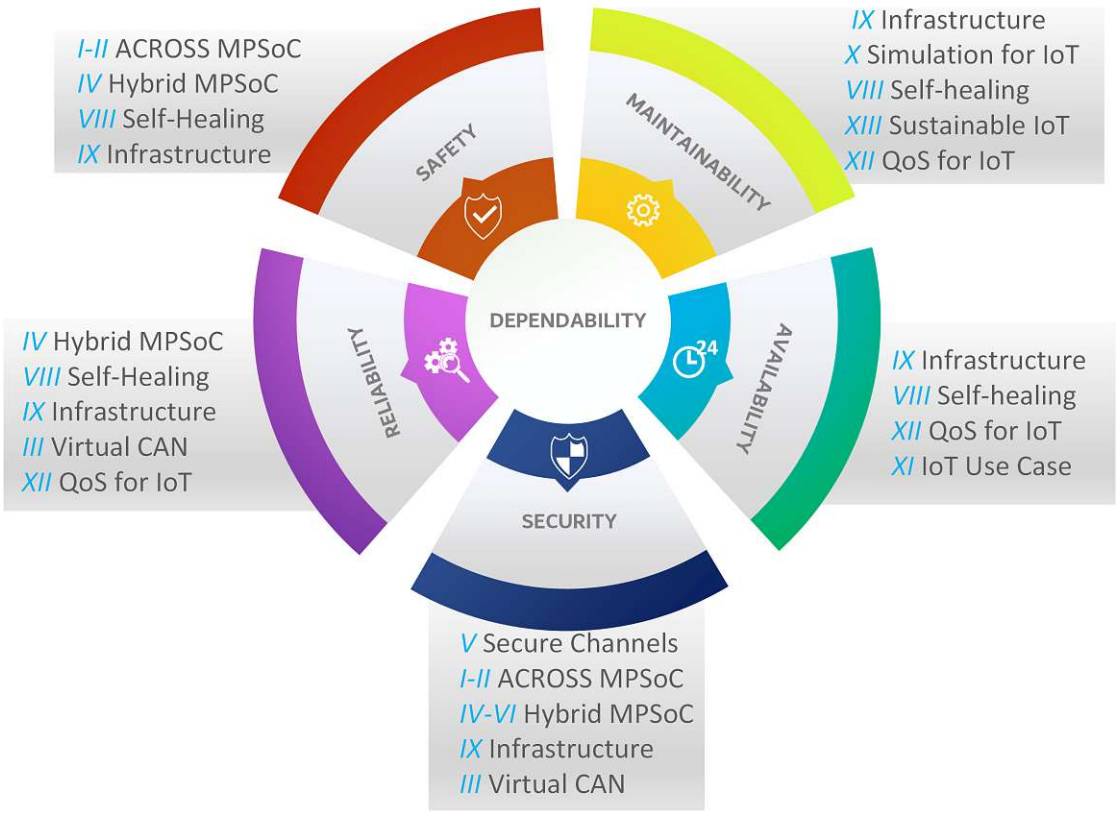


Figure 1.6: The mapping of the publications included in the thesis to the basic dependability attributes.

systems. The standard under designation IEEE P2510 [15] is providing guidelines for establishing quality of data sensor parameters.

In addition to IEEE standards for IoT, there are other activities that are working on regulating CPS/IoT such as the Framework for Cyber-Physical Systems published by NIST [53], or IoT-Enabled Smart City Framework by IES [38].

All these initiatives are working towards a common goal, which is to establish a standardized basis for dependable design and implementation of CPS and IoT applications, which we characterize as CPS/IoT Ecosystem.

What? Standards provide us with normative, guidelines and requirements that if applied will produce a guarantee of achieving certain properties. We are aiming towards CPS/IoT Ecosystems that are conform with standards.

Table 1.2: International standards that support dependable CPS/IoT Ecosystem [27]

Organization	Standard	Domain	Topic
ISO/IEC	29164	Biometrics	Embedded Framework
ISO/IEC	25000:2005	Software Engineering	Software Quality Requirements and Evaluation
ISO/IEC	9496	Embedded Systems	CHILL Language
ISO/IEC	18372	Hardware Design	RapidIO Chip Interconects
ISO/IEC	62061	Industrial machines	Safety Machinery
ISO/IEC	26262	Automotive	Safety
ISO/IEC	9126	Software Engineering	Software Quality
IPC	6011-6018	Circuit Boards	Quality of PCBs
IPC	9151	Circuit Boards	Process Quality
IPC	9194, 9199	Circuit Boards	Statistical Process Control (SPC)
IPC	4811, 4821, 2316, 7091	Embedded Systems	Design and production guidelines for embedded PCB
IPC	7090-7095	Embedded Systems	PCB Packaging
IEC	61508, 61511, 61513	Control systems	Safety, Industrial Systems, Nuclear Industry
IEC	62278	Railway	Safety
IEC	62304	Medical	Safety
IEEE/IEC	62014	System On Chip	Quality of electronics and software
IEEE	1500	Embedded Systems	Testing and integration
IEEE	1687	Embedded Systems	Access and control for instrumentation in semiconductor devices
IEEE	1003	Information Technology	Operating System, Interfaces, Applications
IEEE	1450	Digital Design	Testing Language
IEEE	802	Communication	Protocols, interfaces, synchronization
SEA	J2356	Distributed Systems	Model
SEA	J2640	Embedded Systems	Software Design Automotive
SEA	AS5506	Embedded Real-Time Systems	Architecture Analysis & Design Language (AADL)

1.5.2 Methods

The scientific method consists of a constant investigation of existing scientific theories and publishing of new theories [51]. It requires a continual assessment of testing methods and ongoing formulation of new tests that may provide additional insights on existing knowledge or create new knowledge [5]. In this work we adhered to this approach by starting our research with an evaluation of existing scientific works, creating a hypothetical baseline with applicable assumptions, creating experimental frameworks for testing and evaluation, and finally proposing new theories or augmenting existing ones.

Three predominant methodologies in computing research and related disciplines are: a) Theoretical methodology, which builds on mathematical foundations and explains the computing systems through logical reasoning and proofs; b) Experimental methodology, which relies on empirical evaluation of phenomena through series of tests and observations, c) Computer Simulation methodology, which studies phenomena by creating their respective computer models and simulating their behavior in various environments [51].

Works presented in Chapters 2-8, and other works presented in Section 1.4 are predomi-

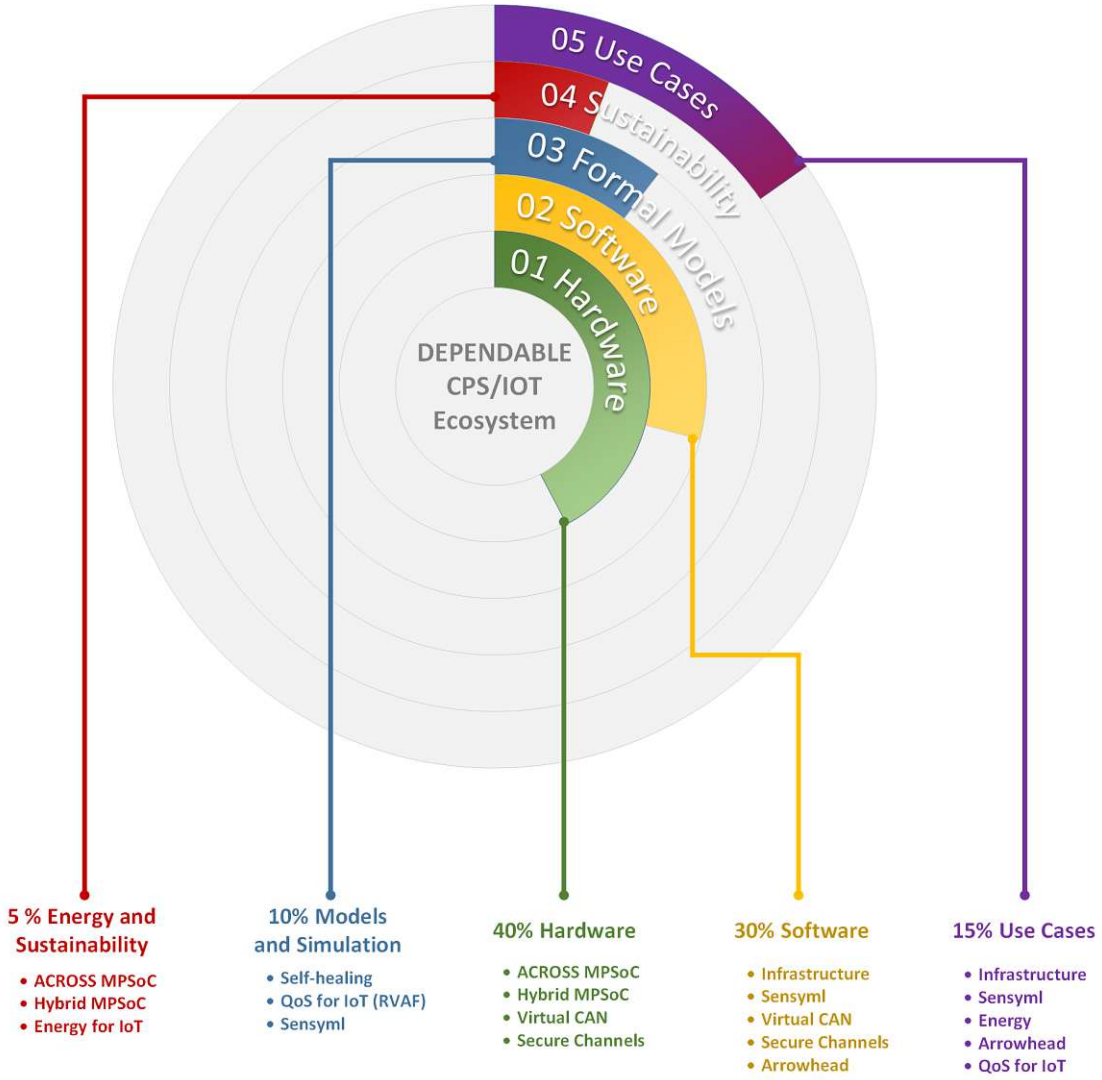


Figure 1.7: An overview of research disciplines within CPSoS in relation to publications included in the thesis.

nantly accomplished by applying experimental methodology. However, as we can observe in Chapters 7, 8 or in publications [88], [47], [86] we combined theoretical methodology with experimental evaluations. In the work proposed in [54] we proposed a simulation environment that strongly supports a computer simulation methodology approach.

Further, we conducted a case study for CPS/IoT Ecosystem based on a real-world application [55]. In addition, we created a survey of innovative hardware technologies for mixed-criticality integration [65], including another important method of scientific research in our research background.

The work presented in this thesis covers five different fields of research within computer science (see Figure 1.7):

- **Hardware** - focusing on research and development of hardware architectures or components, in the extent of 40% of published works.
- **Software** - with a focus on development and evaluation of system software, middleware and applications, estimated to around of 30% work completed.
- **Case Studies and Surveys** - transferring research ideas into real-world scenarios and collecting fundamental information on existing research and technologies as the basis for the works presented in our publications. We estimate the extent of this research at 15%.
- **Models and Simulation** - creating relevant simulation frameworks for CPS/IoT Ecosystems and proposing relevant models to create predictions and assumptions. This covers about 10% of the presented works.
- **Energy and Sustainability** - describes research on energy aware systems and methods for conservation of energy in computing systems. This topic is presented in 5% of the work completed.

Using a standardized methodology and scientific method provides us with generic guidelines on how to organize and execute a research task. However, it doesn't tell us which tools to use to achieve this goal. In the following section we will introduce the most important concepts, frameworks, and tool-sets used in the research for this thesis and their relation to the notion of dependable CPS/IoT Ecosystem.

How? In this section we identify methods that can be used to achieve above defined goals and requirements.

1.5.3 Architectural Concepts and Frameworks

In the following section we will identify architectural concepts that are supporting the notion of dependable CPS/IoT Ecosystem. These systems are as mentioned before highly complex with an increasing trend, so they are built with a high grade of heterogeneity, both vertically across the operation layer spectrum (i.e., cloud, fog, sensor) and also horizontally within each operational layer. This is necessary to achieve the full system comprehension for a certain level of dependability. In the work presented in later chapters, we apply certain state-of-the-art concepts, tools, and methods that are designed to increase the degree of comprehension when building CPS/IoT Ecosystem.

How? In this section we specify architectural concepts, hardware and software frameworks to achieve dependable design, execution and operation of CPS/IoT Ecosystem.

Service-Oriented Architecture (SoA)

A Service-Oriented Architecture (SoA) approach comprises a system design and implementation with a focus on encapsulating functions in services which can be exchanged and applied over common interfaces [83]. An SoA approach supports extended dependability properties such as reconfigurability and interoperability, and can even be aligned with state-of-the-art standards for industrial automation (e.g., IEC 61499) [43]. Finally, an SoA approach is highly flexible, and we can design services from a hardware layer up to the application level as can be seen in the following two examples.

The work we present in Chapters 2, 4, 3, 5 is based on the ACROSS SoA multiple system on chip platform (MPSoC) [88], and earlier work on GENESYS architecture. [80]. They define a set of requirements for dependable distributed embedded systems which can be applied in multiple industrial domains. It propagates abstraction, partitioning, and segmentation as means of reducing complexity. An abstraction considers dividing the system into different integration levels: system, device, chip. A partitioning ensures strict separation between components on each level of abstraction, and a segmentation reduces task complexity by dividing a complex task in a series of simpler jobs. The services can be divided as: a) core services or essential services, b) optional services or system functions that are not essential in every instance, c) application-specific services. The perspective of the system services can be local and confined to a single component, or global and available to the whole ensemble.

Furthermore, the work presented in Chapter 7 is implemented using the Arrowhead framework [46]. The Arrowhead framework is designed for automation of IoT systems and their secure and dependable application in industrial systems. It is conceptualized on the notion of local clouds. Applications and systems are automated using a multi-cloud approach, where each cloud maintains a set of services that can be operated locally or exchanged with other clouds using a secure connection. Each cloud is designed around a set of core mandatory services and application-specific services.

A CPS/IoT Ecosystem is a heterogeneous and complex structure. In order to apply it in systems with strict safety and security constraints we need to increase the explainability of the system and ensure temporal and spatial isolation of components. Use of standardized interfaces as a binding element reduces complexity and provides composability, the ability to add and remove components without custom integration requirements [80].

How? We used SoA architectures to create a basic environment for dependable CPS/IoT Ecosystem. It is an important part for answering Research-questions RQ1 - RQ6.

Hardware Acceleration and MPSoC

A common approach to building cyber-physical systems (CPSs) is a homogeneous hardware platform either with software running on COTS hardware or custom-built dedicated hardware. However, in recent years the trends are evolving towards more heterogeneous and more flexible solutions. This revolution is driven by two major paradigm shifts:

a) Building CPS as a single multidisciplinary process rather than an ambiguous set of independently created systems, and b) Internet-of-Things (IoT), an enormous collection of independent systems operating in isolation or in groups over the internet.

Major challenges behind these paradigm shifts include: "retirement" of single core chips, performance increase for rapid increase in additional functionality, reduction in size, energy consumption, and security. For years now single-core CPUs have been replaced by multi-core variants due to the obvious advantages of increased performance, reduced power consumption, etc. In consumer electronics this change has been adopted easily. However, in industrial applications, especially ones with safety related requirements, this change is much more challenging to implement. Certification of devices without clear separation between different operational units is an extremely difficult task.

Field Programmable Gate Array (FPGA) platforms created a huge impact on server and data center technology already [85]. With hybrid SoC platforms the advantages of FPGA flexibility and performance can be also transferred in other applications ranging from small IoT devices (e.g. ZynqBerry [52]) to applications complex as Advanced Driver Assistance Systems (ADAS) [87][4].

Hardware accelerators are becoming a well-established tool in the scope of CPS/IoT Ecosystem due to their flexibility, and the ability to support both essential dependability requirements and the extended requirements that we defined in Section 1.5.1.

A System-on-chip represents a fully functional system that is contained within a single chip, including all of its core components: processor, memory, and peripherals on-board [50]. This means that we could have a full system working as hardware accelerators or components for dedicated tasks, or with a certain level of dependability.

A hybrid multiple system-on-chip (MPSOC) approach can be observed in Chapters 4, 5, where we enable mixed-criticality integration through separation of functions. The safety functions were performed on an FPGA-based custom-built components, and non-critical functions on a loosely coupled COTS-based system (i.e., ARM).

Further, hardware accelerators can be also an efficient Fog device within a CPS/IoT Architecture, where it can be used as communication hub or data analysis system. We demonstrated how to implement FPGA based re-configurable nodes for adaptive signal processing using a similar approach [62]. Hardware accelerators are used from cloud servers, over network infrastructure, to data analysis in fog. In addition to clear performance advantages they also support energy-sustainable development with clear advantage over CPU or GPU solutions [74], [76].

How? We used hardware accelerators on an FPGA basis and MPSoCs as a platform for implementation of dependable hardware architectures. This is the cornerstone technology for Research-questions RQ1 - RQ4.

Time-Triggered System-on-Chip (TTSoC) Architecture

Dependable systems include safety-critical systems, or systems where failure to properly function can result in catastrophic consequences. Safety-critical systems have strict deadlines for communication and execution of tasks [70]. The time-triggered architecture is a framework for the design and implementation of large dependable deterministic systems [71]. TTSoC represents a component methodology that integrates the design principles of an SoC with a deterministic time-triggered framework [82]. It allows the implementation of distributed systems in a fully deterministic way.

In our work on the ACROSS [47] architecture we demonstrated how a high level of temporal and spatial determinism can be achieved using a TTSoC enabling the integration of multiple safety critical systems or functions on a single chip. The temporal and spatial determinism can be also characterized as an essential feature when building mixed-criticality applications as described in Chapters 4 and 5.

Time-triggered architectures are designed for a high level of composability [71]. This is a property that enables systems to expand and compose into larger systems without any side effects [82]. This allows systems to expand, reduce, and reconfigure such that the eventual faults at the interfaces between individual components are contained.

The time-triggered paradigm is not exactly reserved for highly critical systems and applications. The IEEE 802 standard [12] is constantly expanding and introducing "time-triggered" features. This brings time-sensitive communication into the standard Ethernet communication infrastructure. This will provide a necessary determinism. This will reduce need for proprietary hardware or software and increase overall interoperability. In conclusion, the time-sensitive communication infrastructure is an essential component of a dependable CPS/IoT Ecosystem.

How? Certain types of systems require extreme determinism and the time-triggered communication architectures are designed to facilitate this kind of determinism. TTNoC is an on-chip communication technology that enables solutions for RQ1-RQ4.

Runtime Verification and Monitoring

To establish the confidence in a computational system and increase its credibility, it is necessary to have a process of "checks and balances" in terms of scientific observation and evidence. Runtime Verification as a field of research stems from a discipline in computational science called Verification & Validation (V&V).

V&V are two essential processes when evaluating and quantifying the correctness of a computational artefact. [79]. Verification is "*the process of assessing software correctness and numerical accuracy of the solution to a given mathematical model*". Validation is "*the process of assessing the physical accuracy of a mathematical model based on comparisons between computational results and experimental data*" [79].

V&V methods observe the full spectrum of system executions and corresponding results. This is achieved by either watching the data generated by the solution of the associated

models, or by watching and analysing the experimental results. Runtime Verification (RV) is a subset of V&V, that checks the results of the system's execution against the ones of the model's solution [32]. Sometimes it is referred to as Runtime Monitoring (RM).

RV can be applied to the verification of software components such as functions, services of full programs. Furthermore, it can be used to verify communication protocols and data flow. However, one can also apply the same techniques to verify behavior of full systems such as Fog nodes, industrial machines or vehicles.

CPSs are closely coupled with their respective physical surroundings and as such their execution can be unpredictable to some extent. It is often unfeasible to build a model that can perceive all possible scenarios. Likewise, it is difficult to collect the full spectrum of data necessary to verify all possible execution scenarios. RV provides an alternative approach whereby we observe and verify a single execution at a time, observing relevant variables and checking their correctness at simulation time or on runtime [30].

Information from RV can be used to detect and correct incorrect behavior of a system or its components as presented in [86]. Further, it is a mechanism that can provide guarantees to establish system attributes such as QoS and reach certain level of dependability as shown in Chapter 7.

How? The formal verification of a system is an essential component in its certification. Runtime Verification allows us to evaluate a working system and make sure that its operation is satisfying the system specification. With a high number of extremely heterogeneous components it is difficult to achieve formal verification off-line. Thus, we need to use an on-line verification framework for each individual component.

Dynamic Reconfiguration, Self-healing or Self-adaptation

The ability to tolerate faults and errors, and to resist external influences, is a crucial requirement for safety-critical systems and systems with higher functional constraints. A common approach to fault tolerance is to apply modular redundancy of critical components; this way if one component fails the redundant copies can proceed with the operation [70]. To increase the efficiency of the system one can apply reconfiguration methods that will restore the system or component in case of a fault or an error. Another way to establish fault tolerance is to use implicit redundancies within a system to perform system reconfiguration or adaptation.

In the work we presented in [86] we identified two types of self-healing: structural adaptation and parametric adaptation. In this work we elaborate on the advantages of structural adaptation where a function of a system is restored by rearranging the order of components to replace the faulty ones. In Chapter 7 we present strategies for system reconfiguration that can perform structural and parametric adaptation as well as the combination of two methods.

Self-healing in CPS/IoT Ecosystems provides the ability to tolerate faults without the need for explicit redundancies. The ability of the system to reconfigure and restore its

original function after a fault is an extended attribute of dependability. It also supports other attributes such as availability and sustainability.

How? A system's robustness is often determined by its design. In highly volatile and dynamic systems we are often faced with unpredictable situations that may or may not be defined in the initial design of the system. Thus, the system is not always capable of coping with the emerging faults. Having dynamic reconfiguration mechanisms allows us to adapt to unforeseen scenarios during runtime.

Simulation

Simulation is one of three major research methodologies in computing research [51]. It is the process of understanding or evaluating a system through the observation of its model under the constraints of a simulated environment, where one can observe a system as a whole or as a collection of its components [90]. Computer simulation provides us with the ability to reproduce working conditions of even most complex physical systems in real-time [31]. Studies have shown that the Simulation and model representation of physical systems is one of the main pillars of Industry4.0 [95].

In the work we presented in [54] we demonstrate how simulation can be used to replace a large-scale application environment in CPS/IoT to evaluate among other attributes the scalability of a CPS/IoT Ecosystem implementation.

How? Simulation allows to explore the system dynamics at pre-design or at pre-implementation time. This in turn allows engineers to create computer systems that are better suited for a specific physical system.

Green Computing

Green Computing refers to the ecologically responsible use of computers and their resources, and the design and development of methods that reduce energy consumption and chemical footprint. It also provides guidelines on ecologically sensible disposal of old devices [20]. Green computing initiatives don't affect only consumption but also resource distribution, power management, organization, dimension, and other ways to reduce energy and waste overheads [92].

In Section 1 we presented the future trends in the expansion of CPS/IoT devices, with the tendency for the number of devices to reach 100 billion by 2030. This represents a major overhead on the electrical and environmental infrastructure. The devices used in a CPS/IoT Ecosystem should be therefore designed with a certain level of sustainability, which is not only represented in the energy consumption, but also in the longevity of the devices, and their ability to be serviced and maintained.

For example, the cryptocurrency surge in recent years created a cryptomining industry. The decentralized nature of these activities makes it incredibly difficult to track how much energy they are using. However, models show that the publicly accessible Proof-of-Work block-chains are consuming something in the range of 60 to 125 TWh of electrical energy

per year [89]. The use of energy-inefficient devices for cryptocalculations is creating a major global overhead both in energy consumption and in the population created by it. A similar scenario could be possible with CPS/IoT devices and smart applications if not approached systematically from the beginning.

Chapter 8 introduces an energy consumption model for CPS/IoT Ecosystems as defined in Chapter 6. In addition, we show how alternative design methods such as use of energy harvesting could reduce energy and chemical footprint in large CPS/IoT applications.

How? The rapid increase of computer systems is creating a growing energy and waste footprint in the world. Building energy-efficient and long-lasting computer systems is becoming a necessity, especially if we want to over-populate the world with computer systems creating the so called Smart World.

Development and Operations (DevOps)

DevOps is a concept that bridges the gap between development and operation of software systems. It comprises the automated management of software development, integration, testing, deployment and delivery, monitoring and feedback [68]. It is a widely accepted practice in the world of software engineering, especially for web-based applications [34].

However, DevOps is a relatively new concept for CPS and IoT, where development processes are often ad-hoc and developer-centric. This approach is human-error prone, especially when it comes to integration in large and complex projects. The faults are readily propagated throughout the system, while debugging and testing is often extremely difficult to perform. We consider DevOps to be an essential requirement for CPS/IoT Ecosystems. It reduces development time and human-error potential, and also significantly decreases costs in development and operation.

How? If we look at any model for the development of computer systems or software, e.g., the V-model, we can observe that a major portion of the computer system's life-cycle is in design and development. For CPS/IoT Ecosystems this process is even more complex, as it potentially includes multiple sub-systems, components, and organizations. Having an automated process that supports design, development, testing, and monitoring is essential in order to cope with the complexity of these systems.

1.5.4 Evaluation of Dependability Properties

In previous section, we discussed architectural concepts and frameworks employed in later chapters in order to satisfy the requirements of Section 1.5.1. In Section 1.5.2 we indicated which basic methodologies were used in the scope of the research activities presented in Chapters 2-8. We concluded that the experimental methodology is the most often applied method to evaluate proposed theories and hypothesis.

The dependability of a system can be referred to as its ability to exhibit trustworthiness and continuity of service, and the fact that one can rely on this service in justifiable

and quantifiable manner [73]. Furthermore, dependability is characterized by individual system attributes (see Figure 1.5). The evaluation of these attributes can follow through: a) Verification and validation, or mathematical proofs, b) Testing, by using empirical or probabilistic investigation, and c) Simulation, or model-based inspection. Each attribute is quantified by measurable properties of a system. For example, we can represent reliability as a probabilistic function $R(t) = Pr(T > t) = \int_t^\infty f(x)\,dx$.

Properties such as Mean Time to Failure (MTTF) can be used to as a part of this function to give us a quantifiable measure on how probable is it that a system will fail in a given time frame. To determine whether the system is dependable enough for a certain application, we need to check the conformity of these attributes with the norms and requirements proposed by standards. The evaluation process involves crosschecking functional requirements for the application with the attributes of individual components and requirements set by standards.

For example, in Chapter 7 we proposed the model to formalize QoS for services in an CPS/IoT Ecosystem where we include basic properties to describe QoS requirements such as utilization, deadlines, critically level, and memory requirements. The critical attribute determines the priority of a service compared to other services, higher critically requires higher dependability. Services can require real-time execution, where hard-real time deadlines cannot be missed. An application will set its required values for each individual service, and standards will classify them in a scope of certain domain and with given underlying infrastructure. If an underlying device is incapable of providing a certain level of critically due to architectural shortcomings, it will automatically be propagated to the service or an application in general.

Use of techniques such as segmentation, virtualization, SoA or dynamic reconfiguration, will reduce dependence between individual nodes and components of a CPS/IoT Ecosystem and thus simplify the process of evaluation and certification. We demonstrate some of these approaches in Chapters 4, 5, 7.

How? In this section we reflect on the process of evaluation of certain dependability attributes. It involves multiple methods from experimental evaluation to formal verification. It is a process that is heavily dependant on standards as guidelines for development and operation of these systems.

1.5.5 Challenges and Obstacles

In the text above we described necessary requirements for dependable CPS/IoT Ecosystems. Further, we discussed recommended methodologies and concepts to solve methodical, structural and practical problems. We reflected on methods for evaluation and how to measure and quantify the grade of dependability or reliance in CPS/IoT Ecosystems. However, there are still missing components that need to be built or discovered to make this story complete. Here we would like to highlight some of these problems and give our assessment on the course required to solve them or reduce their effect.

A CPS/IoT Ecosystem is a structural and hierarchical concept built from heterogeneous components, not only in terms of functional objective but also in terms of application domains, scope of operation and level of confidence it needs to provide. In Chapter 5 we presented a brief history of computational systems, and we can observe that CPSs expanded and originated from networked embedded systems, a field of research and development that was present since the beginning of computer systems. They were strongly used in areas requiring high reliance and confidence in a system to deliver a service (e.g., automotive, railway, avionics). CPSs inherit this background and build on top of the infrastructure established in the earlier era.

This means that when we are building a safety-critical CPS application we are not starting from zero, but there exist well established standards and methods we can follow. Also, there is a significant amount of hardware and software produced for this purpose that can be reused. This is especially important for system level software, i.e., device drivers, communication protocols, and operating systems.

The IoT component is also emerging from earlier work performed on different communication paradigms such as wireless sensor networks, mobile networks, or industrial automation. Similarly, as in the case of CPS, the IoT is recycling older technology and making it more focused on the more actual problems.

In other words when building CPS/IoT Ecosystems significant groundwork and materials are ready, and just need to be fitted into a system. This paves the way for auto-generation of CPS/IoT projects, which reduces manual development and potential errors. For this, we are missing an adequate modeling framework that would describe the full-scale stack of CPS/IoT and all operational aspects: hardware platforms, middleware software, application software, networking, data, dependability and operations. There are a number of modeling languages and frameworks (e.g., AutomationML [3], SensorML[18] or Resource Description Framework (RDF) [16]) that describe parts of the system or certain aspects. A modeling framework of this scale would provide a full image of a system, and would enable automated generation of code, configurations, operations pipelines, runtime monitoring and verification. Such a modeling framework would help to increase explainability of CPS/IoT Ecosystems and corresponding applications.

A second major obstacle in the sustainable design and implementation of CPS/IoT Ecosystems is energy consumption and energy conservation. The number of devices is rising each year and there are no clear guidelines on how to implement Green Computing strategies on this scale. It is clear that we need to introduce programming paradigms for CPS/IoT Ecosystems that will conserve energy by design.

One of the major factors when it comes to dependable CPS/IoT Ecosystem is certification. The heterogeneous nature of a CPS/IoT Ecosystem requires adequate approaches towards certification. Certification of software is always a challenging task, but in this case, one needs to check each hardware device, communication protocol and causality between individual components. Some initiatives like the Zephyr RTOS project [22] are working on certification of operating system for IoT. The certification of communication channels

Table 1.3: Representation of Research Questions in Chapters

	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6	RQ7
Virtual CAN	✓	✓✓	✓	✓			
Secure Channels			✓	✓✓			
Hybrid MPSoC	✓✓						✓
Mixed-criticality	✓		✓✓				
CPS/IoT Ecosystem					✓✓		
QoS for IoT					✓	✓✓	
Sustainable IoT							✓✓

✓- Partial ✓✓- Full Support

is handled using existing networking standards. However, a uniform solution or guideline has yet to be proposed.

What? In the previous chapter we define architectural concepts and frameworks to achieve certain dependability aspects in CPS/IoT Ecosystem. However, they are not always enough to reach a required level of reliance and confidence in these systems. Thus, we need to look beyond the state-of-the-art and keep identifying research topics that are important for the goal of dependable CPS/IoT Ecosystems.

1.6 Thesis Synopsis

Chapter 2 (Virtual CAN)

Virtual CAN Lines in an Integrated MPSoC Architecture, Armin Wasicek, Oliver Höftberger, Martin Elshuber, Haris Isakovic, Andreas Fleck, 17th IEEE International Symposium on Object/Component-Oriented Real-Time Distributed Computing (ISORC), Reno, Nevada, USA, pages 158 - 165, 2014.

Chapter 3 (Secure Channels)

Secure Channels in an Integrated MPSoC Architecture, Haris Isakovic, Armin Wasicek, 39th Annual Conference of the IEEE Industrial Electronics Society (IECON), Wien, Austria, pages 4488 - 4493, 2013.

Chapter 4 (Hybrid MPSoC)

A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform, Haris Isakovic, Radu Grosu, 25th IEEE International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, USA, pages 244 - 253, 2016.

Chapter 5 (Mixed-criticality)

A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform, Haris Isakovic, Radu Grosu, "Solutions for Cyber-Physical Systems Ubiquity", IGI Global, Hershey, PA, pages 169 - 194, 2018.

Chapter 6 (CPS/IoT Ecosystem)

CPS/IoT Ecosystem: A Platform for Research and Education, Haris Isakovic, Denise Ratasich, Christian Hirsch, Michael Platzer, Bernhard Wally, Thomas Rausch, Dejan Nickovic, Wilibald Krenn, Gerti Kappel, Schahram Dustdar, Radu Grosu, 14th Workshop on Embeddedand Cyber-Physical Systems Education (WESE 2018), Turin, Italien, Springer International Publishing, 2019.

Chapter 7(QoS for IoT)

QoS for Dynamic Deployment of IoT Services, Haris Isakovic, Luis Lino Ferreira, Irmin Okic, Adam Dukkon, Zlatan Tucakovic and Radu Grosu, IEEE 22. International Conferenceon Industrial Technology (ICIT), Valencia, Spain, 2021. (In Print)

Chapter 8 (Sustainable IoT)

An Energy Sustainable CPS/IoT Ecosystem, Haris Isakovic, Edgar Aaspiazu Crespo and Radu Grosu, EAI Edge IoT, Braga, Portugal, Springer International Publishing, 2020. (In Print)

1.6.1 Contribution of the Author per Chapter

(Virtual CAN) This paper discusses virtualisation techniques for communication channels, demonstrated on the CAN protocol. The author worked on the integration of the virtualization service with he underlying hardware architecture, operating system PikeOs, and the middle-ware framework Autosar. The author provided significant contributions to the practical implementation that led to the publication and participated extensively on the publication itself.

(Hybrid MPSoC) We proposed a novel approach to an earlier implementation of a many-core hardware architecture. The author designed and implemented the new architecture on a novel hardware platform. Further, the author initiated, structured and wrote the publication, with the support of prof. Grosu.

(Secure Channels) In this paper we proposed a security service for System-On-Chip architectures. The author created initial design and implementation of the security service. The proposed architecture was further refined in cooperation with Armin Wasicek. The publication is credited equally to both authors, while the author of this thesis presented the work.

(Mixed-Criticality) In this paper we extended an idea presented in the paper "Hybrid MPsoC" onto mixed-criticality integration and proposed essential system requirements for mixed-criticality integration. The work was co-written by by prof.

Grosu. This publication is an extension of the "Hybrid MPSoC" and thus in the thesis they will be discussed jointly.

(CPS/IoT Ecosystem) The author initiated the publication and proposed an idea of conceptualizing a CPS/IoT Ecosystem as a practical and theoretical structure that will define nature of relationship between CPSs and IoTs. A CPS/IoT Ecosystem as a hierarchical structure supports novel data driven development and operation of distributed computer systems. The author provided major contributions and presented the publication. This is a significant publication for the thesis, because it includes the first definition of a CPS/IoT Ecosystem.

(Sustainable IoT) The author initiated the concept and proposed the model for energy consumption and energy conservation in a CPS/IoT Ecosystem. In addition the author supported the development of the prototype. In conclusion the author also initiated the publication and provided significant contribution in it. Mr. Crespo contributed strongly on the prototype and publication, and prof. Grosu provided valuable insight to various parts of the publication.

(QoS for IoT) We proposed the idea of using DevOps infrastructure as a medium for Runtime Monitoring and Verification, that will enable Quality-of-Service assurance on the scale of a single service and the system in general alike. The author provided major contributions to the work and publication, with prof. Ferreira and prof. Grosu adding significant parts. Irmin Okic, Adam Dukkon and Zlatan Tucakovic were credited for their participation in the practical implementation.

1.7 Summary of the Thesis

A CPS/IoT Ecosystem is a hierarchical structure that unifies a CPS and an IoT in a single organism, with the intention to implement applications with requirements for a high degree of reliance and confidence in the system. As we explained in Section 1.5.5 this is quite a complex task, and it needs to be observed from multiple operational aspects.

In this thesis we aim to present some pieces in this puzzle that we proposed in our earlier works. The research spectrum presented in Section 1.4 develops from hardware related topics at start to run-time verification and green computing towards the end of the list. We will follow the same order (with few exceptions, where however, the same order in terms of research topics is kept). Figure 1.8 shows the chronology of the published work.

1.7.1 Virtual Communication Channels

The cost of electronic components in vehicles increased by 50% in last 10 years, thus it reached 45% of overall production costs in automotive industry [45]. The conventional architecture in automotive systems is centered around an Electronic Control Unit (ECU). We can describe this with a loose statement "one function one ECU". Modern vehicles have around 100 ECUs that are connected via a Control Area Network (CAN). Also, the

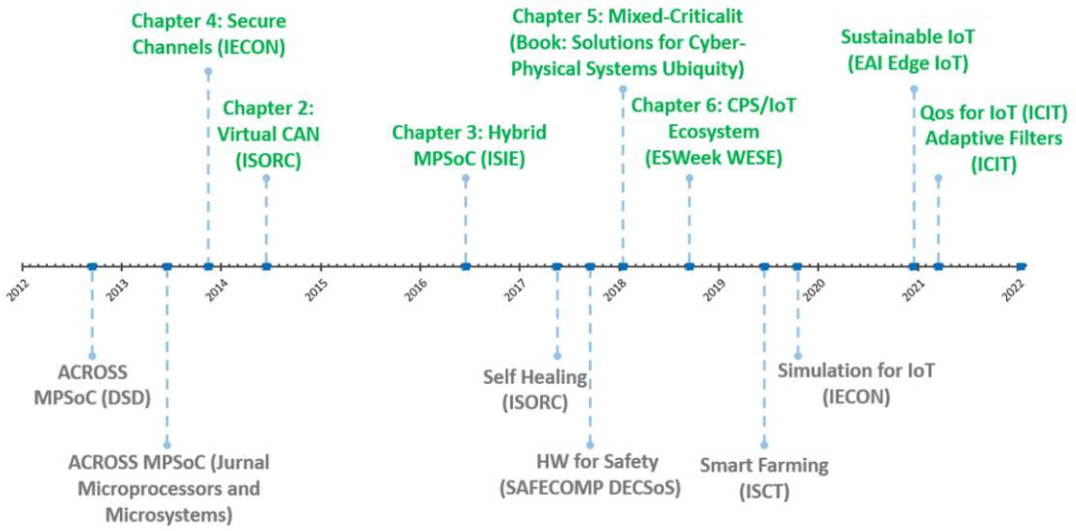


Figure 1.8: Chronological order of research publication. (Green) Publications included in this thesis. (Gray) Supplementary publications.

software systems are getting more complex with millions of lines of code added to the total almost every year, as we can observe in Figures 1.2 and 1.4.

This development method is reaching its physical and economical limits. Manufacturers and regulation bodies aim to produce lighter and more efficient vehicles, and to reduce overall production costs and emissions.

With the introduction of new hardware architectures, e.g., ACROSS [88], Aurix [1], it became feasible to integrate multiple functions on a single ECU without loss of performance or safety properties.

ACROSS MPSoC (see Figure 1.9) is a hardware architecture designed for integration of multiple safety applications on a single chip without interference between individual cores or significant reduction in performance. This architecture removes limitations of conventional single and multi-core architectures. Moreover, it provides full spatial and temporal isolation to all cores.

The Virtual Control Area Network (VCAN) extends the physical CAN interface to all on-chip nodes in a completely transparent manner. It enables access to a physical CAN channel to all on-chip nodes, and the ability to communicate with external nodes without any constraints. Figure 1.10 shows an example with two channels and a multiple of off- and on-chip nodes. ACROSS TTNoC can separate communication between the nodes using time separation. VCAN is using this technology to virtualize physical channels by supplying the individual messages to respective nodes using specific time slots. It supports both BasicCAN and FullCAN, and it can serve applications and CAN devices independent of their CAN infrastructure.

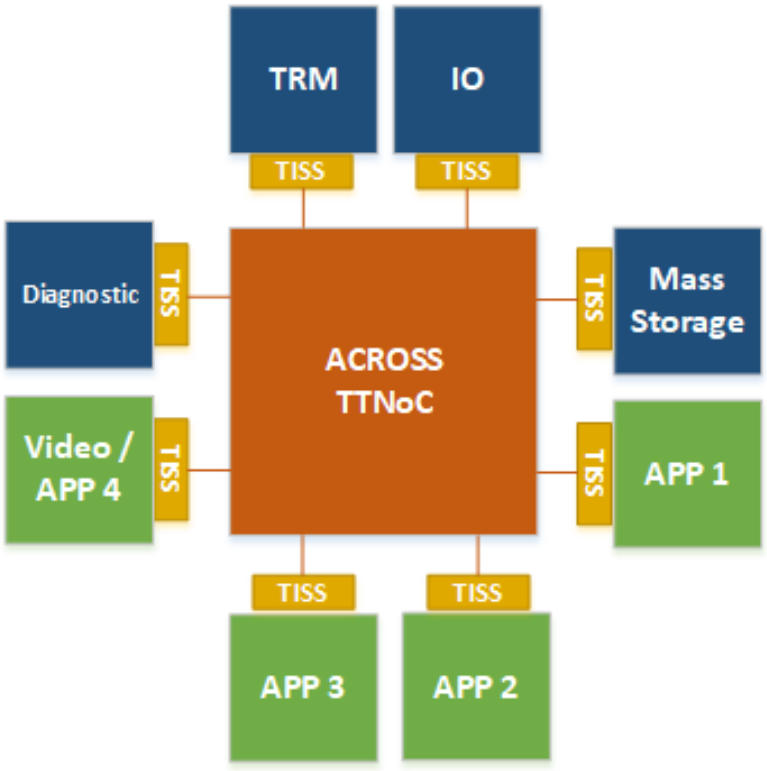


Figure 1.9: ACROSS MPSoC architecture for safety-critical applications. The figure shows a many-core architecture with eight cores. Four cores (blue) are reserved for system/core services and the other four (green) are the cores where applications reside.

The benefits of VCAN are demonstrated on a hybrid engine control use case. It is a prototype for a safety-critical application with fifteen different nodes, five of which are located on the ACROSS MPSoC.

Both technologies introduced in this work are presenting numerous advantages for a CPS/IoT Ecosystem. Application of many-core architectures with safety capabilities will reduce the overall number of physical components in a system. Virtualization of communication channels reduces weight and energy overhead imposed on a system by the weight of cables and other devices used in these networks.

1.7.2 Secure Channels for Integrated MPSoC Architectures

As we mentioned in Section 1.5.1 security is one of the major challenges in the research of modern computing systems. In the scope of an CPS/IoT Ecosystem this problem is significantly more complex to handle due to the heterogeneous nature of such an assembly, hardware and software constraints of the actors, and most important, exposure to the Internet. Thus, it is necessary to approach the security paradigm systematically starting

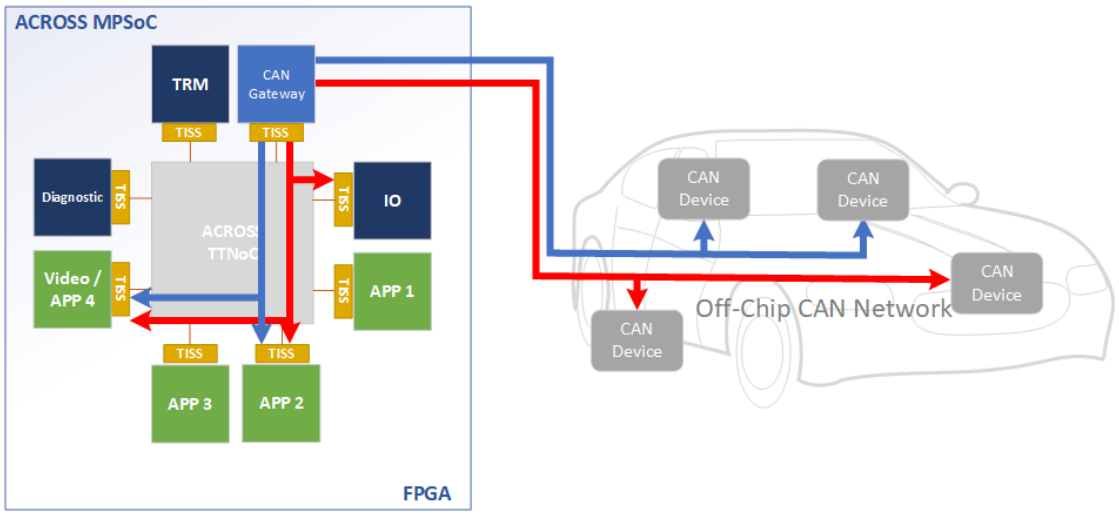


Figure 1.10: VCAN system structure example with two CAN channels.

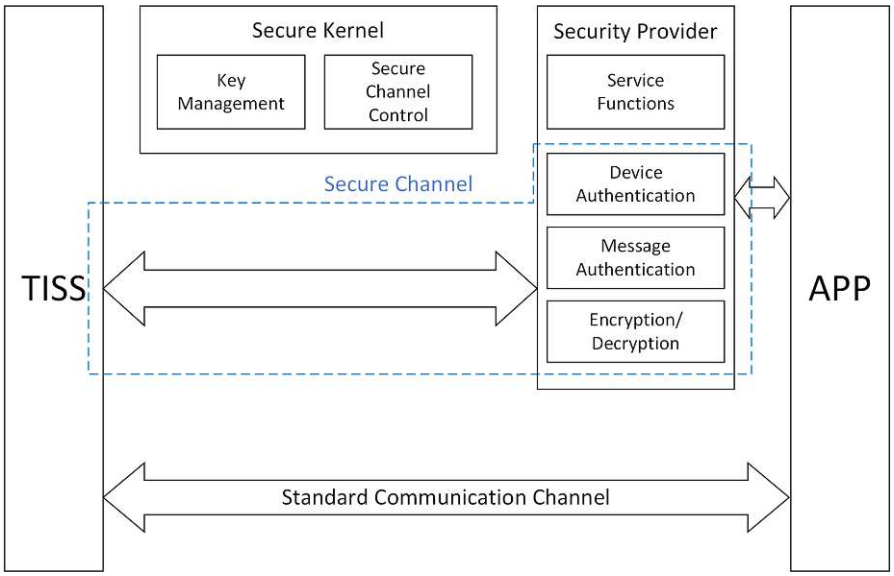


Figure 1.11: Secure communication architecture for an integrated MPSoC architecture.

from hardware architectures to modeling, design and implementation of applications.

This work is an extension of the ACROSS MPSoC and the work on virtual CAN lines presented in the previous section. It proposes a secure communication architecture (see Figure 1.11) for ACROSS MPSoC and similar architectures. The secure communication architecture consists of two basic building blocks, Secure Kernel and Secure Provider. Secure Kernel performs security key management and secure channel control, while Secure Provider performs security services such as encryption, message or device authentication,

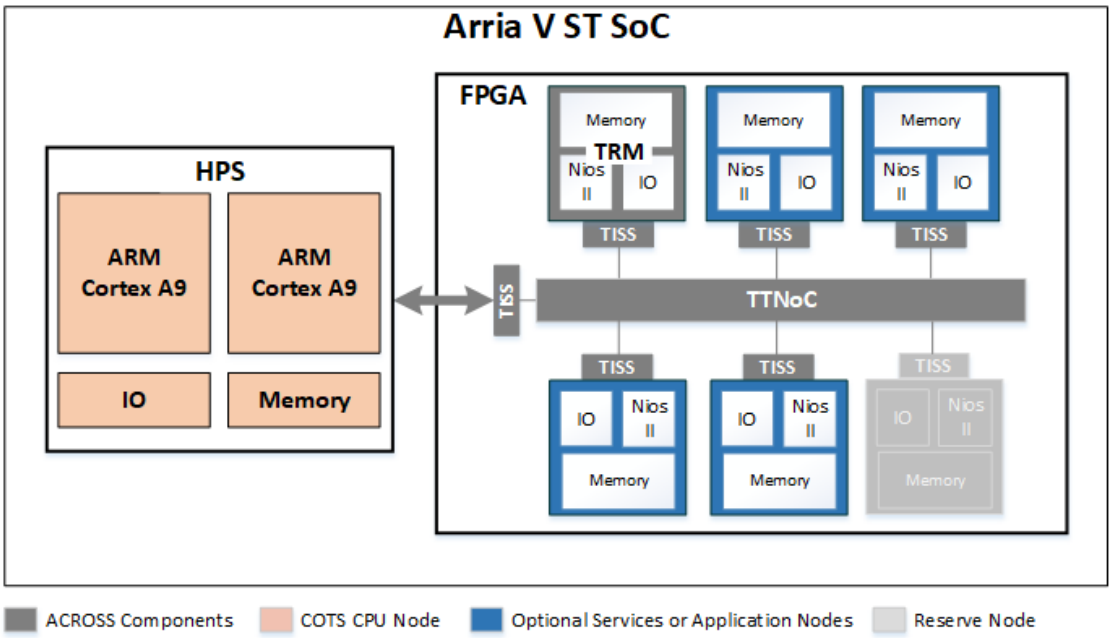


Figure 1.12: A heterogeneous time-triggered architecture on a hybrid SoC platform.

etc. Further, we introduce a notion of Secure Channel, an abstract object composed of regular communication channel and security features on top of this channel.

The proposed security architecture uses underlying features of ACROSS MPSoC such as spatial and temporal isolation of on-chip nodes, integrity of on-chip communication channels manged by Trusted Resource Manager (TRM), and the ability to offer security as a service. Use of underlying infrastructure to enforce security assets allows conservation of resources and integrated security without involvement of application components.

The secure communication architecture presented in Chapter 3 is a collection of design concepts that would conserve resources, enable fine-grained security configuration, and offer security as a service within an integrated MPSoC architecture. These concepts could be observed as solutions for some major security issues in security for CPS/IoT Ecosystem, such as key, user and device management, lack of resources for encryption algorithms, or applications not designed in conformity with security standards.

1.7.3 Hybrid MPSoC and Mixed-Criticality

In this Section we summarize Chapters 4 and 5 as they stem from the same work first presented in the Chapter 4 and then further extended on the more general topic of mixed-criticality in Chapter 5. This work is also an evolution of the work performed in [88] and the works presented in Chapter 2 and 3.

A shortcoming of the ACROSS MPSoC architecture was computing performance, as the

whole MPSoC was implemented on an FPGA. Even with the highest-ranking FPGAs at the time it was limited to eight cores with the maximum core frequency of 400 MHz.

The next generation of FPGAs [60] offered a hybrid solution with a COTS CPU and FPGA integrated on a single chip. This offered new capabilities for architectures such as ACROSS MPSoC. We proposed an extension to ACROSS MPSoC by integrating an on-chip COTS CPU as an ACROSS node (see Figure 1.12). The COTS CPU is represented by a state-of-the-art multi-core CPU with a core frequency above 1GHz. It would be used to offload computationally intensive non-critical tasks from the FGPA nodes. It also increased the number of resources reserved for applications with higher criticality grade that are hosted on the FPGA nodes. This reduced the probability of potential resource-starvation scenarios and increased overall dependability.

This approach followed the principles set by ACROSS MPSoC architecture with slight adjustments to the execution and updated infrastructural IP cores for the implementation of FPGA components and interconnects. The nodes are still separated in space and time and communication is synchronized on the trusted interfaces (TISS).

Mixed-Criticality Integration

In the above section we introduced a heterogeneous MPSoC architecture implemented on a hybrid SoC, that is capable of hosting safety-critical applications and non-critical applications on the same SoC. However, mixed-criticality integration still remains one of the major challenges in research of computing systems. In Chapter 5 we identify historical landmarks for computing research and development that lead to evolution of cyber-physical systems and networked systems in general. Furthermore, we identified main obstacles and challenges towards mixed-critically integration.

CPU architectures have experienced a generational turnover from single-core to multi-core architectures. Due to rapid advancement of general requirements and an increased need for performance, multi-core processors were developed to maximize throughput without major concern for non-interference assurance or deterministic execution of tasks. This created concerning circumstances in safety-related application domains, as the multi-core architectures can not guarantee deterministic execution of safety-critical tasks. Software solutions can approach this goal only so far. The hardware architectures are either designed to prioritize performance or are unable to reveal scheduling algorithms due to extremely high levels of protection on intellectual property. The most common solution is to execute cores in complete isolation, where only one core would function at a time.

This is where solutions such as ACROSS MPSoC or Hybrid SoC presented in Chapter 4 provide obvious advantages. Not only to ensure interference free execution of tasks of a same application, but also to ensure execution of whole independent systems in parallel.

We also identified the lack of a hierarchical or taxonomic approach towards defining the level or grade of system criticality. Often systems are over-dimensioned due to the inability to determine exact requirements. As we discussed in Section 1.5.1, requirements

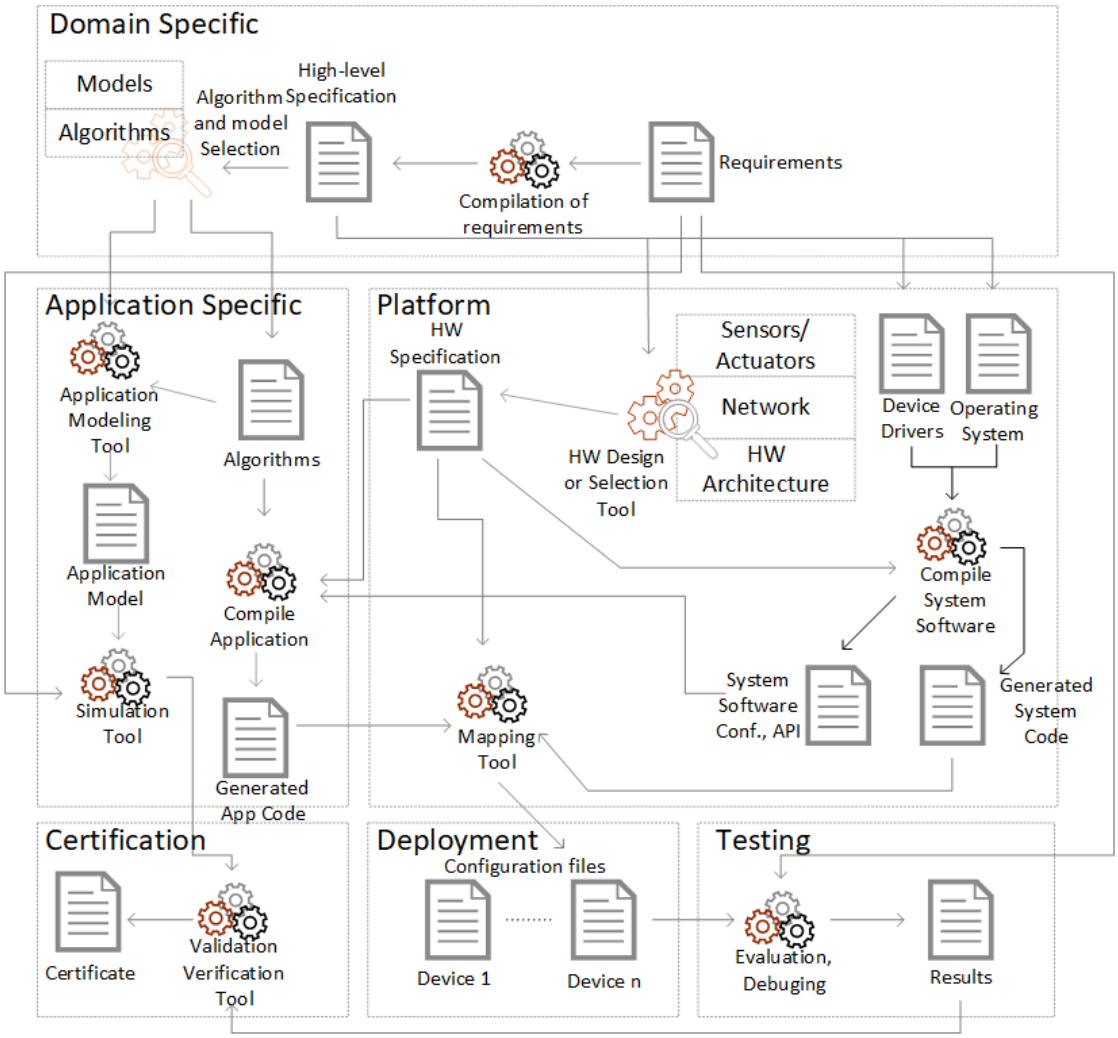


Figure 1.13: A model for integrated development and deployment of CPS.

definition is a multi-dimensional task and in the case when there are no clear guidelines these definitions can overlap or be incomplete.

One of the challenges for mixed-integration is also discussed in Section 1.7.1 and Chapter 2, and that is the inability to manage progressively increasing complexity of CPSs. In Figure 1.13 we show a simple model for integrated development and deployment in CPSs. Managing this process means understanding the system from design to implementation, and being able to verify its properties to ensure that functional and non-functional requirements are met.

In Chapter 5 we also address the followinf question: What is the state of mixed-criticality integration in networked cyber-physical systems or in computing research in general?

In other words, this work laid foundations for the definition of CPS/IoT Ecosystems presented in later Chapter 6.

1.7.4 CPS/IoT Ecosystem: Definition

The idea of a CPS/IOT Ecosystem originated through the project with focus on building an infrastructure necessary to teach and research CPS/IoT topics. A common approach to the research on CPS/IoT is to use simulation data or small-use case scenarios, that fail to reflect the grand scale that is a CPS/IoT in the real world. The project CPS/IoT Ecosystem was conceptualized with a focus on building a real-world infrastructure, starting from the cloud servers, laboratories for fog and edge computing, and large sensor/actuator networks that would mimic the equivalent size real-world applications.

What started as a name for a project soon became a concept describing the correlation between a CPS and an IoT. Not only a collection of tools and devices that would be used to perform experiments, but rather a set of methods and guidelines that would allow us to formally define each component and its role in this system-of-systems. From the start of the project it was obvious that there were no distinct guidelines on how to establish an infrastructure for a CPS/IoT with the objective to host applications that require higher level of reliance and confidence in the system underneath.

In Chapter 1.1 we define a CPS/IoT Ecosystem as a concept and describe its basic attributes. Main characteristic of each CPS/IoT Ecosystem is that it can be divided in three basic scopes of operation: the cloud, the fog, and the swarm. A combination of these three layers of operations allows us to systematically collect data on an enormous scale, transform this data in real-time into control sequences, or use it to learn new emerging insights into systems under observation and optimize their overall performance.

To demonstrate functional and non-functional requirements for CPS/IoT Ecosystems we presented two use cases that depict the sensitivity and complexity of different scopes of operation and application development stages.

The first application, Smart Parking, is a typical application in the family of large-scale Smart City applications. In a city such as Vienna there are over 100,000 parking places [54]. It is a multi-actor application that is deployed over all three layers of operations. Sensors are collecting occupancy data from parking spaces and Fog gateways are filtering and aggregating this data into information which is then stored in the Cloud and forwarded to an end-user or to another application via the Internet.

The second is a Smart Farming application: Smart Vineyards. It is an equally challenging application, similar in scale to the first, with even stricter requirements in terms of accessibility, energy consumption, and data accountability. For example, agricultural settings such as vineyards require particular physical structures as support, and these cause interference with wireless networks; thus a special type of communication infrastructure is required to equip these structures with sensors or actuators. Further, they are often located in remote locations without access to an electrical grid, which means that the systems deployed in these environments need to be self-sustainable.

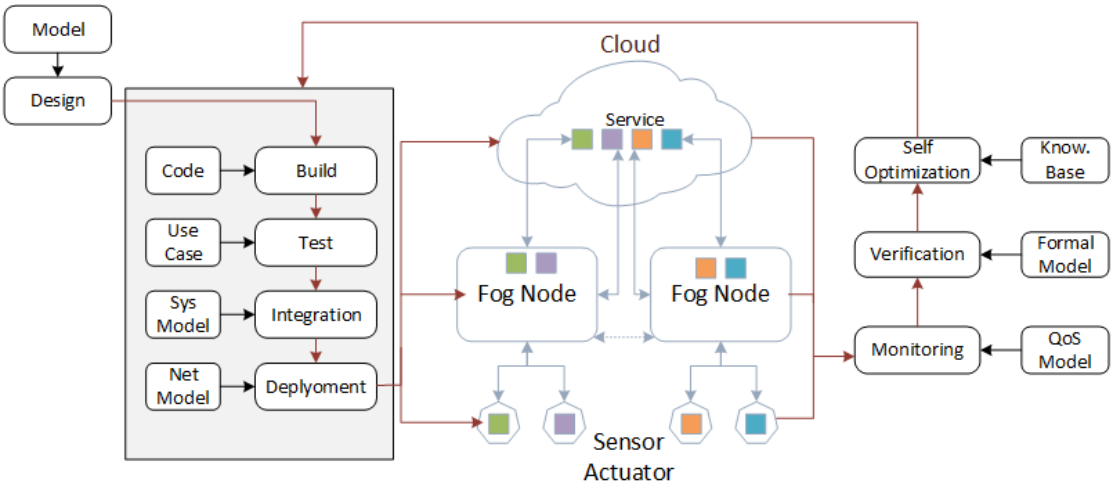


Figure 1.14: A representation of a typical CPS/IoT application with a process of development, deployment and verification, distributed on multiple services and over three scopes of operation.

With this work, we opened a whole pallet of topics that are still vaguely discussed or not discussed at all. Moreover, to expand this concept to build a stable ecosystem, especially for applications, it is necessary to understand their individual components, the interaction between them, and to establish accountability between development and operation.

1.7.5 QoS, DevOps and Runtime Verification

In the previous section, we described the motivation for CPS/IoT Ecosystems as a hierarchical structure with multiple scopes of operation. We consider this structure to be composed of heterogeneous devices and software components. A CPS/IoT Ecosystem is an infrastructure for smart applications. These applications use services from all three scopes of operation, and they are written by one or more individuals or organizations. In the Smart City example, we witnessed how relatively simple applications could be extremely complex due to scale and physical constraints.

Thus, it is necessary to apply different means to manage complexity throughout the development and operation lifecycle of these systems. In Section 1.5.3 we identified SoA as one of the most effective tools to manage complexity, composability and interoperability in large-scale systems. But even with the structural observability that SoA provides, it is extremely difficult to establish a functional workflow for simultaneous development of services, software and hardware components, while maintaining full tractability of errors, and enabling continuous evolution of the system in general.

In terms of software engineering for enterprise and web development, SoA is combined with continuous integration and deployment methods (CI/CD) to achieve fully automated operational lifecycle of each software artifact. CPS/IoT applications have strong roots in

embedded systems that used a more direct developer-centric development approach. For small applications this is still an acceptable method; however for large and extremely large applications with multiple actors and organizations this becomes unfeasible.

In Chapter 7 we define a CI/CD system for an Arrowhead IoT framework. Its purpose is to enable building, integration, deploying and monitoring services or other related artifacts in a full automated manner. With the ability to monitor each service we can establish health status of individual components or a system in general.

On top of the CI/CD infrastructure we proposed a runtime verification extension for the Arrowhead Framework (RVAF) that can monitor and orchestrate Quality-of-Service (QoS) on a scale of a service, node and even system in general. Moreover, the approach can be easily generalized to other service-oriented architectures. It demonstrates how important and useful is a complete CI/CD infrastructure for CPS/IoT Ecosystems.

First, we formalized the problem and proposed a system model that can describe individual nodes and services. Based on this model, we proposed a formalization of QoS that can be used as a QoS specification for a service or a node. Then we defined runtime-adaptation algorithms that would ensure QoS guaranties during operation. The runtime-adaptation strategy is arbitrary, it depends on the CI/CD capabilities and application requirements. Basic runtime-adaptation strategies can be defined as: a) redeploying a service on the same node, b) migration of a service, c) roll-back of service to earlier version and redeployment, d) service termination, and e) resource reallocation for a service.

We implemented an experimental evaluation of RVAF on an actual CPS/IoT infrastructure created in the scope of the project. We can perform the experimentation on the state-of-the-art Cloud and both general purpose single-board computers and industrial grade computers as Fog devices. This is important for the dependability question, as we can evaluate timing factors for each of the CI/CD stages required for complete realization of RVAF. As we implement the CI/CD infrastructure on existing tools, it is important to evaluate the extent of this COTS solution in terms of response times and usability.

1.7.6 Sustainable CPS/IoT Ecosystem

In Chapter 8 we evaluated the sustainability of CPS/IoT Ecosystems and the methods for building self-powered energy harvesting sensor nodes. Throughout this thesis we pointed out that the ongoing computational evolution will yield billions of new devices that will use existing methods for hardware and software design. The question is whether the existing design paradigms are sustainable enough?

To offer a glimpse of the answer to this question we proposed an energy consumption model that can be used to approximate the energy consumption of CPS/IoT devices. In the model we excluded cloud servers as they have a slightly different consumption model. However, use estimates from other authors show the possible energy consumption of cloud servers.

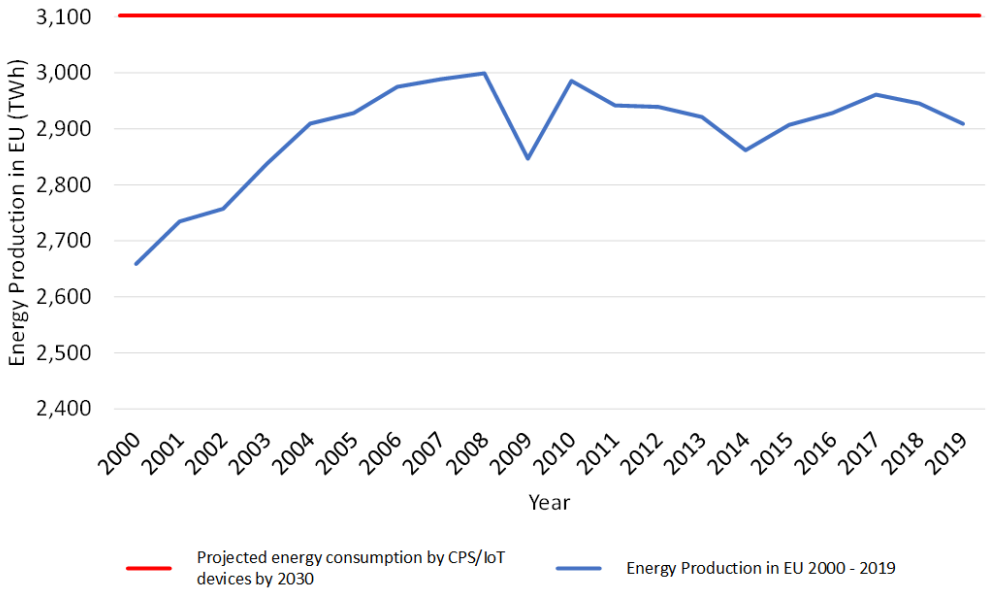


Figure 1.15: Energy production of EU from 2000-2019 according to Eurostat [48]. End the projected energy consumption of CPS/IoT by 2030.

$$E_{IoT} = (E_{IoT_a} \times AR + E_{IoT_p} \times (1 - AR)) \times T \times D_{IoT} \times (\frac{100\% - ER\%}{100})^n \quad (1.1)$$

The model in Equation 1.1 gives an estimate of the energy consumption based on a consumption of an arbitrary device in passive and active mode of operation (E_{IoT_a}, E_{IoT_p}), a ratio between active and passive operation mode (AR), the projected number of CPS/IoT devices in the projection period (D_{IoT}), projected growth of energy efficiency for devices (ER), number of hours in operation over a period of one year (T), and the projection period in years (n).

Further, we explore alternative sources of energy consumption. In particular, we evaluate energy harvesting to provide electrical power to CPS/IoT sensor nodes, without the need for long-term energy storage units. The proposed solution offers a long-term sensor node for remote and inaccessible places that can be used to detect natural catastrophic events such as wildfires, floods, avalanches etc.

The experiments showed that this method is quite feasible to power sensor nodes. Furthermore, it requires no chemical energy-storage device, which makes it sustainable for a longer period of time and is environmentally friendly. Either through changes in design and implementation paradigms, proper modeling approaches or using alternative sources of energy the concept of "green" and sustainable CPS/IoT Ecosystems are essential requirements for present and future applications.

1.8 Contributions of the Thesis

The main objective of this thesis is to provide an insight into the changing landscape of CPSoSs and the emergence of new hierarchical concepts such as a CPS/IoT Ecosystem, and to establish an advisory pathway towards dependable design and implementation of these systems. It gives an overview of the significant publications by the author, their relation to individual research topics, and their role in enabling dependable CPS/IoT Ecosystems. We propose a number of models, methods, tools, and frameworks, that would enable various aspects of dependability in CPS/IoT Ecosystmes. This work summarizes these activities and provides the reader with a broader picture on how they contribute to the improvement of individual dependability properties, and the dependability of a CPS/IoT Ecosystem in general.

We provided side notes intended to sort the contributions in the thesis on Why? defining motivation, What? to point out a goal or requirement, and How? the procedure or the approach to achieve certain contribution. We defined the focus of the thesis using seven research questions:

RQ1: How to build a dependable hardware architectures for CPS/IoT Ecosystem?
RQ2: How to achieve virtualization of hardware resources in time?
RQ3: How to enable dependable mixed-criticality integration in many-core architectures?
RQ4: How to achieve a secure communication architecture on integrated MPSoC?
RQ5: How to conceptualize synergies between CPS and IoT?
RQ6: How to ensure QoS using run-time verification on a scale of CPS/IoT Ecosystem?
RQ7: how to achieve a sustainable CPS/IoT Ecosystem?

The contributions of the thesis are either direct answers to these questions or support the answer in some capacity. We singled out nine individual contributions that can be seen as significant for the topic of dependable CPS/IoT Ecosystems.

Contribution 1: Definition of a CPS/IoT Ecosystem (RQ5)

Why? CPS/IoT infrastructures are currently developed in an ad-hoc fashion. By creating a model or unified specification, it would allow a more systematic and explainable development, deployment, and maintenance of these applications.
What? Define a semi-formal hierarchical structure with clear scopes of operation and interfaces between them.
How? One of the fundamental contributions of this thesis is the definition of CPS/IoT Ecosystems in Chapter 6. We identified the basic building blocks and the requirements for the dependable realization of a CPS/IoT infrastructure, middleware, and applications. This work is complemented by in [54] where we proposed a simulation framework for CPS/IoT Ecosystems that provides capability of simulating sensors and environmental functions, that can be connected to real infrastructure. The contribution corresponds directly to *RQ5*, and it supports *RQ6* and *RQ7* by extension.

Contribution 2: Many-core hardware MPSoC architecture for integrated safety-critical applications (RQ1)

Why? Computer architectures evolve constantly with a focus on increasing performance and reducing power consumption. The switch from single-core to multi-core created a vacuum in the domain of safety-critical systems. Multi-core CPUs are not able to run with the required grade of dependability for safety applications.
What? Hardware architecture or software solution that would enforce and guaranty deterministic execution of tasks.
How? In *RQ1* we address the problem of safety in multi-core architectures. This question has been answered first with the ACROSS MPSoC architecture [88], and then further extended in other subsequent publications [47], [65]. Finally in Chapters 4 and 5, we expand the architecture with hard-coded CPU cores designed to overcome the potential performance issues of the original architecture.

Contribution 3: Mixed-criticality integration (RQ3)

Why? Consumer-driven development of CPS/IoTs often leads to the necessity of combining safety-critical applications or tasks with non-critical tasks. For example, an on-board computer in a vehicle controls media streaming and car health status of a car.
What? A Hardware or a software framework that would allow execution tasks in complete isolation.
How? Contribution 2 creates a baseline for integration of applications with different security and safety requirements on a single MPSoC. In Chapter 5 we define requirements for mixed-criticality, and demonstrate capability of the architecture proposed in 4 to enforce them. These contributions directly correspond to RQ3, and also significantly support RQ1 and RQ7.

Contribution 4: Secure communication architecture for dependable systems (RQ4)

Why? The number of computer systems in CPS/IoT applications like automotive applications or railway is constantly increasing. They are reaching a physical maximum of how many electronic systems they can integrate.
What? The challenge is to increase the number of electronic or computer systems while maintaining physical limitations. In this particular case, we want to extend limited communication interfaces so they can be used by multiple nodes at the same time.
How? Using ACROSS MPSoC we are able to virtualize any physical channel on a level of TTNoC. A physical CAN channel would be separated in multiple time slots for each core on the MPSoC, thus allowing all of the cores to access the communication channel in a completely transparent way. This work corresponds directly to RQ2, and partialy to RQ3 and RQ4.

Contribution 5: Virtualization of communication channels on an integrated MPSoC architecture (RQ2)

Why? The number of computer systems in CPS/IoT applications like automotive applications or railway is constantly increasing. They are roaching a physical maximum on how many electronic systems they can integrate.
What? The challenge is to increase the number of electronic or computer systems while maintaining the physical limitations. In this particular case, we want to extend the limited communication interfaces so they can be used by multiple nodes at the same time.
How? Using ACROSS MPSoC we are able to virtualize any physical channel on a level of TTNoC. A physical CAN channel would be separated in multiple time slots for each core on the MPSoC, thus allowing all of the cores to access the communication channel in a completely transparent way. This work corresponds directly to RQ2, and partially to RQ3 and RQ4.

Contribution 6: Model and proof of concept of CI/CD on the Arrowhead framework (RQ6)

Why? A CPS/IoT Ecosystem is conceptualized as a large-scale collection of heterogeneous devices and software components. These components can be developed and operated by multiple individuals and organizations. We need to ensure that all components are properly integrated, that they could be updated if needed, and in case of error that we can trace back where and why this error occurred. Using manual integration, development and deployment this is almost impossible, especially on large scale projects.
What? Integration of CI/CD with CPS/IoT requires integration on different scales of operation. The basic example would include build, integration, deployment, and monitoring stages as automated tasks in the development process.
How? CI/CD and DevOps in general originated in large scale software engineering projects and web applications. Using existing tools we implemented a CI/CD pipeline that is able to orchestrate above mentioned basic stages of development in a fully automated way. This contribution offers a partial answer to the RQ6. and also supports RQ5.

Contribution 7: QoS orchestration using RV (RQ6)

Why? In Section 1.5.3 we explained how dynamic reconfiguration of a system can increase dependability and robustness of a system during runtime.
What? To ensure dependability and robustness through self-adaptation, we need to be able to monitor components, services, functions or even individual variables. Further, it is necessary to establish a formal definition or specification of a system that can be verified in an automated manner. Finally, we need to be able to perform corrective measures on the system to correct faults and errors.
How? In Chapter 7 we showed how to use exiting CI/CD infrastructures to perform all tasks listed in *What?*. This enables us to monitor variables of choice, and based on the data to decide what corrective measure to perform. The corrective measures are defined

by reconfiguration algorithm that specifies various strategies that can be applied. Some examples include redeployment, rollback, termination etc. This provides a direct answer to RQ6 and even can be considered part of the answer for RQ7.

Contribution 8: Energy consumption model for CPS/IoT Ecosystem (RQ7)

Why? As mentioned above, it is expected that the number of CPS/IoT devices will reach 100 billion by 2030. This represents a technological and socioeconomic revolution, where we are entering the era of a Smart World. However, one of the vaguely answered questions is how sustainable is this revolution?
What? Devise a mathematical model where we could approximate the energy consumption of CPS/IoT devices in a specific time period.
How? The proposed model uses a consumer electronic power consumption model. It calculates power consumption based on a number of devices, projection time period, time of use over a year, average energy consumption per device in active and passive state of operation, and yearly efficiency increase ratio. This way we can forecast the energy consumption using different scenarios and evaluate what is a most probable scenario based on geopolitical, economic, and technological conditions.

Contribution 9: Alternative power source for CPS/IoT sensor nodes using energy harvesting from a surrounding environment (RQ7)

Why? CPS/IoT applications can be deployed on remote and inaccessible places. Some of the applications can be considered safety critical (e.g., wildfire, or avalanche detection sensors). Having a constant power supply is difficult in such cases. Also, the sheer number of CPS/IoT devices will create an enormous overhead on existing power grids. Building CPS/IoT devices using alternative power sources is thus a necessity to achieve a sustainable CPS/IoT Ecosystem.
What? Identify potentially useful alternative energy sources and demonstrate their effectiveness for the given use case.
How? Energy harvesting is not a new method to supply computer systems. We designed a sensor node for the detection of wildfires that is operating without battery and is powered using thermo-electric generators (TEG). The device uses a narrow-band communication network, and it is active only in case it detects an event, in this case wildfire. It can stay dormant for a number of years, but in case of a fire event it would provide a burst of information including the location and outside conditions.

1.9 Future Work

Creating dependable CPS/IoTs requires deterministic behavior in hardware and software. This is a multi-stage process that starts with the definition of requirements. Verification of these requirements is a challenging task in CPS/IoT Ecosystems due to the heterogeneity and scale of these systems. Further, requirements are written in a natural language and cannot be processed by the verification algorithm without translation.

1.9.1 Formalization of Requirements

Work is ongoing to formalize requirements via artificial intelligence (AI). For extremely large systems, the manual translation of requirements in formal languages is time-consuming and almost invisible task. Recent advancement in natural language processing (NLP) technology allows us to build translators from natural language into arbitrary formal language. This would have an enormous impact on the verification of CPS/IoT applications. It would allow verification and feasibility checks at design time before any code is yet written. Further, it opens the door for the automated generation of runtime monitors, allowing the systematic observation of large numbers of signals or variables with minimum overhead. In Chapter 7 we demonstrated a direct relationship between runtime verification and service dependability. The next step is to fully automate this process from requirements to a system model, runtime monitoring, and self-adaptation.

1.9.2 Reference Model for CPS/IoT Ecosystem

Earlier we discussed how CPS/IoT applications share large portions of infrastructural code and design patterns with embedded systems, web applications, and other fields. We are working on a reference model for CPS/IoTs that would allow the construction of the necessary infrastructure and create the layout for an application. Moreover, the model would describe the basic configuration of the infrastructure, e.g., virtual machines in the cloud, fog nodes, middleware software etc. We would then use the model to generate configurations, code and service templates for all the scopes of operation. This would remove the possibility of human error in configurations of devices and system software, and provide recommended configuration according to guidelines set by requirements or standards. This approach has also other advantages such as rapid prototyping and fine-grained configuration of properties relevant to dependability of a system.

1.9.3 Dependable DevOps

The role of DevOps is essential when building and maintaining large systems. As we mentioned earlier DevOps tools are based on the requirements from other fields. It is necessary to evaluate these tools for the use in dependable systems, or extend them by adding additional features to a component. In the ADEPTNESS [2] we propose a service-oriented architecture that is enabling dependable development and operations for CPS/IoT Ecosystems. The architecture would consider all DevOps stages starting with build and integration, auto-generation of tests, deployment and delivery of artifacts, monitoring and verification, and uncertainty detection and self-adaptation.

1.9.4 Mixed-Reality Simulation Environment

Dependable CPS/IoT applications are often difficult to evaluate in a real-world environment due to safety restrictions and limitations. Digital twins provide us with the infrastructure to evaluate work in progress without violating any safety guidelines. In an ongoing work we propose a mixed-reality simulation environment for evaluating mobile

robotic platforms. This allows us to evaluate and verify the behaviour and operation scenarios of mobile robots by combining physical actors with simulated actors. The system is integrated with a runtime verification component, that enables the observation and verification of system properties during these scenarios.

1.10 Conclusion

Current trends in science, technology, and society in general, are heading towards a *Smart World*, a concept where ubiquitous computing and machine intelligence is integrated into physical objects (e.g., consumer electronics, wearable), virtual resources (e.g., cloud services), social paradigms (e.g., social networks), business and industry (e.g., factory and business digitization) and even human reasoning [75]. These systems are becoming entangled in a collaborative network of devices, services and socioeconomic practices that create more sustainable ecosystems [96]. They are also a way to improve legacy methods and routines, and to assert sufficient confidence in technology, such that is is allowed to completely govern certain aspects of human existence.

Recently, we witnessed how a sequence of geopolitical events can lead to a global crisis in production of electronic systems. The COVID-19 pandemic, together with political shifts, created a vacuum in chip production and supply [24]. In effect, it stalled the production of electronic components and systems all over the world, shutting down factories and resulting in financial crisis on a global scale.

If we look over the horizon, in ten to fifteen years, we will have a global network of smart devices that will interact with each other in an autonomous manner and connect everything from personal every day devices to vehicles and industrial systems. Events like the chip-shortage or crypto-energy crisis could create a major disturbance in the operation of companies and society in general.

In Chapter 1 we introduced general requirements, standards, and methods for a dependable CPS/IoT Ecosystem. We identified state-of-the-art architectural concepts and frameworks that provide functional and operational infrastructure to ensure essential dependability properties. Creating a dependable system according to traditional dependability models is more difficult if we want to incorporate emerging trends such as green computing and self-adaptation. Thus, we expanded the model with additional attributes such as sustainability or reconfigurability as these requirements are stepping out from the domain of the optional and becoming essential for a number of CPS/IoT applications. A CPS/IoT Ecosystem is extremely heterogeneous in all aspects of development and operation. There is an enormous number of technologies, tools and frameworks being developed at the moment. Finding a reference model as a bottom line for integration of these components into ecosystems is necessary if we want to ensure interoperability and traceability. We describe some of these concepts and their functional advantages towards dependability of CPS/IoT Ecosystem.

In the later chapters we discuss particular topics that are explored separately in different

works but are unified around the concept of CPS/IoT Ecosystem. In Chapter 2 we explore virtualization in the domain of communication channels and hardware interfaces using time division to distribute messages from a single physical channel to multiple virtual addresses. This work was extended on a secure communication architecture in Chapter 3 where we use integrated properties of the underlying architecture to assert security measures and create a security service completely transparent to the application. Chapters 4 and 5 explore the notion of hybrid MPSoC architectures and their ability to perform mixed-criticality integration. We continue with Chapter 6 where we define the concept of CPS/IoT Ecosystem and its basic building blocks. Implementation of CI/CD and its application to achieve perform runtime verification and Quality of Service orchestration is presented in 7. Finally, in Chapter 8, we explore sustainability of CPS/IoT Ecosystems and alternative methods of power supply for its devices.

A CPS/IoT Ecosystem is a living organism; it evolves over time, assimilating new technologies and constantly expanding its abilities. To compress all different aspects necessary to explain CPS/IoT Ecosystems in a single document would be borderline impossible. With this brief overview and relevant examples from hardware, software, security, operations, runtime verification, and energy efficiency we have provided a snapshot of the CPS/IoT Ecosystem concept at its current stage of evolution. We hope this thesis will support and inspire future works and contribute to the common goal of better comprehension of CPS, IoT, and any emerging ideas that may be born by combining these two concepts.

1.11 Bibliography

[1] 32-bit AURIX™ TriCore™ Microcontroller - Infineon Technologies. https://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-microcontroller/.

[2] ADEPTNESS Project. https://adeptness.eu/.

[3] AutomationML. http://www.automationml.org/o.red.c/home.html.

[4] Automotive FPGA Applications - Intel® FPGAs. https://www.intel.com/content/www/de/de/automotive/products/programmable/applications.html.

[5] Definition of SCIENTIFIC METHOD by Oxford Dictionary. https://www.oxfordreference.com/view/10.1093/oi/authority.20110803100447727.

[6] European telecomincations standards institute (ETSI). https://www.etsi.org/.

[7] IEEE 2413-2019 - IEEE Standard for an Architectural Framework for the Internet of Things (IoT). https://standards.ieee.org/standard/2413-2019.html.

[8] Industry4.0 Platform. https://www.plattform-i40.de/PI40/Navigation/EN/Home/home.html.

[9] International electrotechnical comission (IEC). https://www.iec.ch.

[10] International Organization for Standardization (ISO). https://www.iso.org/home.html.

[11] ITU: Committed to connecting the world. https://www.itu.int:443/en/Pages/default.aspx.

[12] LMSC, LAN/MAN Standards Committee (Project 802). https://ieee802.org/.

[13] National Institute of Standards and Technology. https://www.nist.gov/.

[14] P1451-99 - Standard for Harmonization of Internet of Things (IoT) Devices and Systems. https://standards.ieee.org/project/1451-99.html.

[15] P2510 - Standard for Establishing Quality of Data Sensor Parameters in the Internet of Things Environment. https://standards.ieee.org/project/2510.html.

[16] RDF - Semantic Web Standards. https://www.w3.org/RDF/.

[17] SAE International. https://www.sae.org/site/.

[18] Sensor Model Language (SensorML) | OGC. https://www.ogc.org/standards/sensorml.

[19] The IEEE Standards Association - Home. https://standards.ieee.org/.

[20] What is Green Computing? - Definition from Techopedia. http://www.techopedia.com/definition/14753/green-computing.

[21] World Wide Web Consortium (W3C). https://www.w3.org/.

[22] Zephyr Project RTOS. https://www.zephyrproject.org, Feb. 2019.

[23] IPC International. https://www.ipc.org/, May 2020.

[24] Why Shortages of a $1 Chip Sparked Crisis in Global Economy. *Bloomberg.com*, Apr. 2021.

[25] U. Ahmad. QoS Architectures: A Detailed Review. *International Journal of Reviews in Computing (IJRIC)*, 11:32–47, Sept. 2012.

[26] M. Albano, P. M. Barbosa, J. Silva, R. Duarte, L. L. Ferreira, and J. Delsing. Quality of service on the arrowhead framework. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, pages 1–8, May 2017. ISSN: null.

[27] S. Ali, T. Al Balushi, Z. Nadir, and O. K. Hussain. Standards for CPS. In S. Ali, T. Al Balushi, Z. Nadir, and O. K. Hussain, editors, *Cyber Security for Cyber Physical Systems*, Studies in Computational Intelligence, pages 161–174. Springer International Publishing, Cham, 2018.

[28] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan. 2004.

[29] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad. Proposed embedded security framework for Internet of Things (IoT). In *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*, pages 1–5, Feb. 2011.

[30] E. Bartocci. Monitoring, Learning and Control of Cyber-Physical Systems with STL (Tutorial). In C. Colombo and M. Leucker, editors, *Runtime Verification*, Lecture Notes in Computer Science, pages 35–42, Cham, 2018. Springer International Publishing.

[31] E. Bartocci, E. M. Cherry, J. Glimm, R. Grosu, S. A. Smolka, and F. H. Fenton. Toward real-time simulation of cardiac dynamics. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*, CMSB '11, pages 103–112, New York, NY, USA, Sept. 2011. Association for Computing Machinery.

[32] E. Bartocci and Y. Falcone, editors. *Lectures on Runtime Verification: Introductory and Advanced Topics*. Programming and Software Engineering. Springer International Publishing, 2018.

[33] S. Baruah, H. Li, and L. Stougie. Towards the Design of Certifiable Mixed-criticality Systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22, Apr. 2010. ISSN: 1545-3421.

[34] M. D. Bayser, L. Azevedo, and R. F. G. Cerqueira. ResearchOps: The case for DevOps in scientific applications. *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[35] C. Becker, S. Betz, R. Chitchyan, L. Duboc, S. M. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters. Requirements: The Key to Sustainability. *IEEE Software*, 33(1):56–65, Jan. 2016. Conference Name: IEEE Software.

[36] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters. Sustainability Design and Software: The Karlskrona Manifesto. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 467–476, May 2015. ISSN: 1558-1225.

[37] T. Bécsi, S. Aradi, and P. Gáspár. Security issues and vulnerabilities in connected car systems. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 477–482, June 2015.

[38] M. J. Burns. IES-City Framework. page 150, 2018.

[39] G. Callou, P. Maciel, E. Tavares, E. Sousa, B. Silva, J. Figueiredo, C. Araujo, F. S. Magnani, and F. Neves. Sustainability and dependability evaluation on data center architectures. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 398–403, Oct. 2011.

[40] L. Chazette and K. Schneider. Explainability as a non-functional requirement: Challenges and recommendations. *Requirements Engineering*, 25(4):493–514, Dec. 2020.

[41] L. Chen, M. Ali Babar, and B. Nuseibeh. Characterizing Architecturally Significant Requirements. *IEEE Software*, 30(2):38–45, Mar. 2013. Conference Name: IEEE Software.

[42] Cisco. QoS Frequently Asked Questions. https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/22833-qos-faq.html, 2021.

[43] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin. Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability. *IEEE Transactions on Industrial Informatics*, 11(3):771–781, June 2015. Conference Name: IEEE Transactions on Industrial Informatics.

[44] Deloitte. Software is transforming the automotive world. *Deloitte Insights*.

[45] Deloitte. Semiconductors – the Next Wave Opportunities and winning strategies for semiconductor companies. Technical report, Deloitte, 2019.

[46] J. Delsing. *IoT Automation: Arrowhead Framework*. Taylor & Francis Inc, Boca Raton, Feb. 2017.

[47] C. El Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek. The ACROSS MPSoC – A new generation of multi-core processors designed for safety–critical embedded systems. *Microprocessors and Microsystems*, 37(8, Part C):1020–1032, Nov. 2013.

[48] European Commission. Eurostat. https://ec.europa.eu/eurostat/web/main/home.

[49] K. Fowler. Dependability [reliability]. *IEEE Instrumentation Measurement Magazine*, 8(4):55–58, Oct. 2005.

[50] S. B. Furber. *ARM System-on-chip Architecture*. Pearson Education, 2000. Google-Books-ID: J_Fu_YTVD9gC.

[51] E. A. Gamukama and O. Popov. The Level of Scientific Methods Use in Computing Research Programs. pages 176–183, 2008.

[52] T. E. GmbH. "ZynqBerry" Modul mit Xilinx Zynq-7010 in Raspberry Pi Formfaktor. https://shop.trenz-electronic.de/de/TE0726-03M-ZynqBerry-Modul-mit-Xilinx-Zynq-7010-in-Raspberry-Pi-Formfaktor.

[53] E. R. Griffor, C. Greer, D. A. Wollman, and M. J. Burns. Framework for cyber-physical systems: volume 1, overview. Technical Report NIST SP 1500-201, National Institute of Standards and Technology, Gaithersburg, MD, June 2017.

[54] I. Haris, V. Bisanovic, B. Wally, T. Rausch, D. Ratasich, S. Dustdar, G. Kappel, and R. Grosu. Sensyml: Simulation Environment for large-scale IoT Applications. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 3024–3030, Oct. 2019.

[55] I. Haris, A. Fasching, L. Punzenberger, and R. Grosu. CPS/IoT Ecosystem: Indoor Vertical Farming System. In *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pages 47–52, June 2019.

[56] I. Haris, L. Lino Ferreira, I. Okic, A. Dukkon, Z. Tucakovic, and R. Grosu. QoS for Dynamic Deployment of IoT Services. In *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, pages 1144–1151, Mar. 2021.

[57] R. M. Hasani. An Automated Auto-encoder Correlation-based Health-Monitoring and Prognostic Method for Machine Bearings. *CoRR*, abs/1703.06272, 2017.

[58] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 426–437, Sept. 2016.

[59] A. Humayed, J. Lin, F. Li, and B. Luo. Cyber-Physical Systems Security—A Survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, Dec. 2017. Conference Name: IEEE Internet of Things Journal.

[60] Intel. Intel® Arria® Series FPGAs and SoC FPGA - Intel® FPGA. https://www.intel.com/content/www/de/de/products/details/fpga/arria.html.

[61] H. Isakovic, E. A. Crespo, and R. Grosu. An energy sustainable cps/iot ecosystem. In S. Paiva, S. I. Lopes, R. Zitouni, N. Gupta, S. F. Lopes, and T. Yonezawa, editors, *Science and Technologies for Smart Cities*, pages 305–322, Cham, 2021. Springer International Publishing.

[62] H. Isakovic, S. Dangl, Z. Tucakovic, and R. Grosu. Adaptive Signal Filtering Platform for a CPS/IoT Ecosystem. In *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, pages 1391–1396, Mar. 2021.

[63] H. Isakovic and R. Grosu. A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform. In *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, pages 244–253, June 2016.

[64] H. Isakovic and R. Grosu. A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform. In *Solutions for Cyber-Physical Systems Ubiquity*, pages 169–194. IGI Global, 2018.

[65] H. Isakovic, R. Grosu, D. Ratasich, J. Kadlec, Z. Pohl, S. Kerrison, K. Georgiou, K. Eder, N. Druml, L. Tadros, F. Christensen, E. Wheatley, B. Farkas, R. Meyer, and M. Berekovic. A Survey of Hardware Technologies for Mixed-Critical Integration Explored in the Project $$EMC^2$$. In S. Tonetta, E. Schoitsch, and F. Bitsch, editors, *Computer Safety, Reliability, and Security*, Lecture Notes in Computer Science, pages 127–140, Cham, 2017. Springer International Publishing.

[66] H. Isakovic, D. Ratasich, C. Hirsch, M. Platzer, B. Wally, T. Rausch, D. Nickovic, W. Krenn, G. Kappel, S. Dustdar, and R. Grosu. CPS/IoT Ecosystem: A Platform for Research and Education. In R. Chamberlain, W. Taha, and M. Törngren, editors, *Cyber Physical Systems. Model-Based Design*, Lecture Notes in Computer Science, pages 206–213, Cham, 2019. Springer International Publishing.

[67] H. Isakovic and A. Wasicek. Secure channels in an integrated MPSoC architecture. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pages 4488–4493, Nov. 2013.

[68] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer. What is DevOps? A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, pages 1–11, New York, NY, USA, May 2016. Association for Computing Machinery.

[69] H. S. Kang, J. Y. Lee, S. Choi, H. Kim, J. H. Park, J. Y. Son, B. H. Kim, and S. D. Noh. Smart manufacturing: Past research, present findings, and future directions. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 3(1):111–128, Jan. 2016.

[70] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Real-Time Systems Series. Springer US, second edition, 2011.

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

[72] J. Laprie. Resilience for the Scalability of Dependability. In *Fourth IEEE International Symposium on Network Computing and Applications*, pages 5–6, July 2005.

[73] J.-C. Laprie. Dependability Evaluation of Software Systems in Operation. *IEEE Transactions on Software Engineering*, SE-10(6):701–714, Nov. 1984. Conference Name: IEEE Transactions on Software Engineering.

[74] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren. A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. *arXiv:1702.06392 [cs]*, June 2017. arXiv: 1702.06392.

[75] H. Liu, H. Ning, Q. Mu, Y. Zheng, J. Zeng, L. T. Yang, R. Huang, and J. Ma. A review of the smart world. *Future Generation Computer Systems*, 96:678–691, July 2019.

[76] J. Monson, M. Wirthlin, and B. L. Hutchings. Implementing high-performance, low-power FPGA-based optical flow accelerators in C. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 363–369, June 2013. ISSN: 2160-052X.

[77] S. Nasiri, F. Sadoughi, M. H. Tadayon, and A. Dehnad. Security Requirements of Internet of Things-Based Healthcare System: a Survey Study. *Acta Informatica Medica*, 27(4):253–258, Dec. 2019.

[78] D. K. Nilsson and U. E. Larson. Secure Firmware Updates over the Air in Intelligent Vehicles. In *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, pages 380–384, May 2008.

[79] W. L. Oberkampf and C. J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, Cambridge, 2010.

[80] R. Obermaisser and H. Kopetz. *GENESYS: An ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken, Aug. 2009.

[81] T. Olzak, J. Sabovik, J. Boomer, and R. M. Keefer. *Microsoft Virtualization: Master Microsoft Server, Desktop, Application, and Presentation Virtualization*. Syngress, June 2010. Google-Books-ID: J0LD00xWLH4C.

[82] C. P. Paukovits. *The time-triggered system-on-chip architecture*. Thesis, 2008. Accepted: 2020-06-30T18:05:37Z.

[83] J. Phelps and B. Busby. Service-Oriented Architecture–What Is It, and How Do We Get One? *Educause Quarterly*, 30(3):56–58, 2007.

[84] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice-Hall, Upper Saddle River, NJ, 1996.

[85] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope. *A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services*. June 2014.

[86] D. Ratasich, O. Höftberger, H. Isakovic, M. Shafique, and R. Grosu. A Self-Healing Framework for Building Resilient Cyber-Physical Systems. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 133–140, May 2017.

[87] J. Rettkowski, A. Boutros, and D. Göhringer. Real-time pedestrian detection on a xilinx zynq using the HOG algorithm. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, Dec. 2015.

[88] C. E. Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek. The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems. In *2012 15th Euromicro Conference on Digital System Design*, pages 105–113, Sept. 2012.

[89] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller. The Energy Consumption of Blockchain Technology: Beyond Myth. *Business & Information Systems Engineering*, 62(6):599–608, Dec. 2020.

[90] R. E. Shannon. *Systems Simulation: The Art and Science*. Prentice-Hall, 1975. Google-Books-ID: cWpRAAAAMAAJ.

[91] A. Singh. An Introduction to Virtualization. `https://webx.ubi.pt/~hgil/utils/Virtualization_Introduction.html`, 2004.

[92] S. Vikram. Green computing. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 767–772, Oct. 2015.

[93] A. Wasicek, O. Höftberger, M. Elshuber, H. Isakovic, and A. Fleck. Virtual CAN Lines in an Integrated MPSoC Architecture. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 158–165, June 2014.

[94] W. Weber, A. Hoess, J. van Deventer, F. Oppenheimer, R. Ernst, A. Kostrzewa, P. Doré, T. Goubier, H. Isakovic, N. Druml, E. Wuchner, D. Schneider, E. Schoitsch, E. Armengaud, T. Söderqvist, M. Traversone, S. Uhrig, J. C. Pérez-Cortés, S. Saez, J. Kuusela, M. van Helvoort, X. Cai, B. Nordmoen, G. Y. Paulsen, H. P. Dahle, M. Geissel, J. Salecker, and P. Tummeltshammer. The EMC2 Project on Embedded Microcontrollers: Technical Progress after Two Years. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 524–531, Aug. 2016.

[95] A. Wortmann, O. Barais, B. Combemale, and M. Wimmer. Modeling languages in Industry 4.0: an extended systematic mapping study. *Software and Systems Modeling*, 19(1):67–94, Jan. 2020.

[96] C. Zhu, V. C. M. Leung, L. Shu, and E. C.-H. Ngai. Green Internet of Things for Smart World. *IEEE Access*, 3:2151–2162, 2015. Conference Name: IEEE Access.

CHAPTER 2

Virtual CAN Lines in an Integrated MPSoC Architecture

Virtual CAN Lines in an Integrated MPSoC Architecture

Armin Wasicek[1,2],
University of California, Berkeley[1]
EECS Department
Email: arminw@berkeley.edu

Oliver Höftberger[2], Martin Elshuber[2], Haris Isakovic[2], Andreas Fleck[3]
Vienna University of Technology
Institute for Computer Engineering[2]
Institute for Mechanical Engineering and Mechatronics[3]
Email: {oliver,martine,haris}@vmars.tuwien.ac.at

***Abstract*—The standard solution for automotive control networks is the Control Area Network (CAN) bus. Almost any vehicular computer system comprehends at least one CAN line. For the past two decades, software development for control system has been strongly connected to the properties and interfaces of the CAN bus. Currently, the automotive industry is in the middle of a technology leap towards an information-based industry. New technologies are getting ready to fulfill newly emerging requirements for innovative products such as hybrid engine control, intelligent energy management, and advanced driver assistance systems. Integrated Multi-Processor-on-a-Chips (MPSoCs) will be one part of the solution to provide an adequate computing infrastructure for these newly emerging systems. The established technologies like the CAN bus will have to be reconsidered. In this work, we propose a virtual CAN overlay that abstracts the communication interfaces of an MPSoC to provide the Application Programmer Interface (API) of CAN to programmers. The overlay provides the standard behavior of a CAN line and works transparently over chip boundaries. The major implications is that the programmers can continue their used software development approaches and tools when introducing a new computing infrastructure. The main benefit is that the productivity can be maintained during this critical phase. In summary, our solution helps to mitigate the effects from a technology shift to integrated MPSoCs. Our approach is fully compliant with new automotive software development approaches like AUTOSAR.**

I. Introduction

The automotive industry is undergoing a time of substantial change: current technologies and business models are are getting obsolete and new ones have to be researched. The newly emerging automotive marketplace will be fueled by innovation. It is expected that electronics and software will make up 75% of all future innovations [1]. The percentage of production cost of electronics (including software) increased from 19% in 2004 to 40% in 2010 and it will eventually reach 50% in 2020 [2]. Cars will be future software platforms, much like telephones became nowadays.

Traditional, federate automotive system architectures have been recognized as a bottleneck: in a federated architecture each system function is implemented as a separate computer system (i.e., an Electrical Control Unit (ECU) in automotive terms). A subsystem is then formed by several loosely coupled ECUs (e.g., through a CAN bus). Clearly, the obedience to physical units restricts automotive system architectures greatly. For instance, current vehicles already employ up to 100 ECUs, which is the upper limit to the number of devices a car can harness [3]. Moreover, adding more and more ECUs is a significant cost driver. The solution to overcome these limits is to integrate several system functions in single ECUs, thus forming an *integrated architecture* [4]. Integrated architectures that implement the required partitioning mechanisms can be realized either in software (e.g., as a hypervisor) or in hardware (e.g., as an MPSoC). In this work we focus on MPSoC-based integrated architectures, because we are convinced that they are appropriate to solve the partitioning challenge for hard real-time systems appropriately [5].

Keeping pace with the rapid technological development is challenging, time-to-market is a critical success factor for any future product based on software. An automotive software development process is a complex process that requires teams of engineers with different domain expertise to collaborate, it comprehends a myriad of different tools, diverging standards, and varying legal regulations for each of the global six key markets[1], etc. Introducing new technologies in this tightly organized process and changes happen slowly and incremental. For instance, one of the recent milestones in automotive software engineering is the introduction of the AUTomotive Open System ARchitecture (AUTOSAR)[2], whose project plan was released in May 2003 and the first cars with AUTOSAR technology inside were launched in the market in 2008. Setting up appropriate tool chains and development methodologies simply takes time.

The main contribution of this paper is a virtual CAN overlay to speed up the integration of new computer platforms leveraging MPSoC technologies. The presented virtual CAN overlay facilitates similar properties and behavior to a real CAN bus. Implementing automotive applications against this virtual CAN interface leverages existing software development methodologies and tool chains while introducing a new computational MPSoC platform.

The paper is organized as follows: Section II elaborates on integrated MPSoC architectures which form the basis for our research. Section III contains a technical description of the proposed virtual CAN overlay. In Section IV we present a case study of a hybrid engine controller which has been implemented and tested in an automotive Hardware in the Loop (HiL) testbed. Finally, we discuss some related work in Section V and draw a conclusion in Section VI.

[1] I.e., China, India, Western Europe, Japan, Korea, and the United States
[2] http://www.autosar.org/

II. An Integrated MPSoC Architecture

In this section we present briefly the ACROSS MPSoC architecture which is a research platform for hard real-time embedded systems. We the following sections we discuss implementation and validation of the virtual CAN overlay on this platform. Our results, however, are transferable between similar system architectures with predictable interconnect and according spatial and temporal partitioning properties.

A. Benefits of Integrated Architectures

In a federated architecture, the provision of dedicated resources for each subsystem sums up to a high amount of redundant hardware. Integrated architectures aim at reducing this overhead by providing common computing resources to several system functions and thus *virtualizing* the required resources for executing a single system function. These architectures have the potential to enable massive cost savings, reliability improvements, and to overcome limitations for spare components and redundancy management [4].

For instance, integrated architectures can help to reduce the wiring of an automotive system by replacing physical cabling with virtual channels. This directly leads to lower manufacturing costs since fewer connectors, cables, electronic parts, and assembly steps on the production line are required. Furthermore, a reduction of the number of connectors can lead to improved reliability of the entire system, because, if fewer connectors are used, one major source (i.e., 30 %) of electrical failures in cars gets eliminated.

B. The rise of MPSoCs

A MPSoC incorporates multiple, potentially heterogeneous processing cores and other functional units on a single silicon die. Compared to general-purpose single core processors, MPSoCs can provide enormous computational capacity in an energy-efficient and cost-efficient way. The roadmaps of the semiconductor industry [5] show a very clear trend towards multi-core technology, and we can safely assume that the majority of future high- end processors will be MPSoCs. Today, MPSoCs are typically applied in personal computers or consumer electronic devices like smart phones or tablets.

Safety-critical systems are systems whose failure could result in loss of life, significant property damage, or damage to the environment. Examples are flight control systems for aircraft, automotive control systems, medical devices, industrial control systems or nuclear power plants. MPSoCs could bring many benefits (e.g., energy and area efficiency, increased computational performance, specialized cores, reduction of physical units) to safety-critical applications. However, the currently existing MPSoC architectures were not designed with a strong focus on safety and certification and thus have serious drawbacks and limitations for hard real-time applications.

C. The ACROSS MPSoC in a nutshell

The ACROSS MPSoC is a computer system architecture specifically targeted at safety-critical applications, i.e., systems including safety functions that have to fulfill the highest certification requirements. The ACROSS project[3] has been started in order to overcome the limitations of MPSoCs for hard real-time systems. This is achieved by implementing spatial and temporal partitioning (also called segregation, or non-interference) between single components.

[3]http:www.across-project.eu

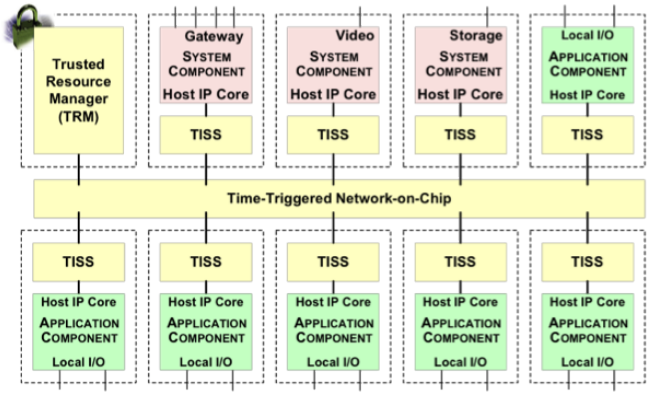


Fig. 1. ACROSS MPSoC architecture

An ACROSS MPSoC (see Figure 1) consists of two subsystems. The application-specific subsystem (green and red) encompasses a set of potentially heterogeneous components and is targeted to implement the actual function of an application. The trusted subsystem (yellow) provides an interconnect to integrate application components into a system. The core innovation of the ACROSS MPSoC is the Time-Triggered Network-on-a-Chip (TTNoC) that enables the dependable interconnection of IP cores (i.e., components) [6]. Each core in the MPSoC has access to a consistent chip-wide notion of time (i.e., the macro tick).

1) Structure of the Interconnect: The trusted subsystem provides a set of core services towards the application-specific subsystem. These core services manifest themselves in the Trusted Interface Subsystem (TISS) which essentially implements a periodic and sporadic message transfer scheme, as well as some services to control the attached component. The trusted subsystem acts autonomously and transfers messages according to an a priory defined communication schedule between the sender and the set of receivers (multi-cast is supported) at a message's defined send instance with reference to the macro tick. An immediate benefit for the system engineer is that the time-triggered schedule is free of conflicts, so no online arbitration required and temporal behavior is 'designed', so no wearisome analysis after the implementation has to be done. Moreover, the trusted subsystem will ensure fault-containment [7] by exploiting information about the permitted temporal behavior of the host IP cores in order to detect and contain an arbitrary temporal failure of a host (e.g., babbling idiot, masquerading faults). For this purpose, each TISS acts as a guardian of the hosted component preventing temporal and spatial interference.

2) Anatomy of a Component: The application-specific subsystem is further enhanced by deploying a combination of middleware and system components to customize the generic core services to a particular application domain. System components (red) usually are used to implement a specific gateway functionality that can be shared by several application components (green). Middleware layers refine single

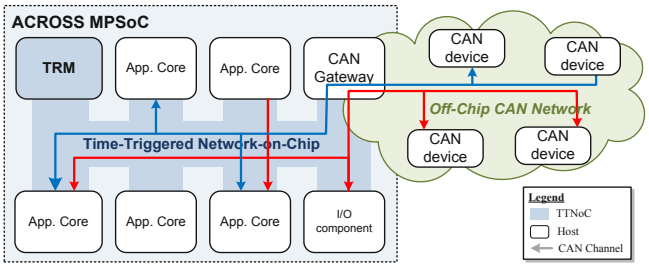


Fig. 2. VCAN overlay system structure

application components. For instance, a port of the PikeOS microkernel [8] is available to provide typical Real-Time Operating System (RTOS) services (e.g., real-time scheduling) to application components. Single application component run distinct instances of PikeOS and communicate only using the core services. A refinement of the core services towards the automotive domain is the virtual CAN overlay which is also implemented as a Middleware according to ACROSS terminology.

D. Connection to AUTOSAR

The PikeOS microkernel has the capability to implement different APIs, so-called personalities. For instance, a personality for AUTOSAR is available. This personality gives the the application programmer the possibility to implement against the domain-specific Runtime Environment (RTE) interface rather that towards the native system call interface. On the bottom of the middleware stack, however, there is a missing link between the microkernel and the TISS. This link can be established by either directly integrating the core services as a new Complex Device Driver which requires additional integration effort on the application programmer's side. Or AUTOSAR middleware and TISS can be linked via the CAN overlay. This solution does not require any changes to the application programmer's programming model or tool chain[4].

III. Virtual CAN lines

The Virtual Control Area Network (VCAN) overlay provides VCAN lines on top of a MPSoC. It enables the seamless integration of physical and VCAN lines. The application programmer just has to deal with nodes in a CAN network. The location (i.e., either on-chip or off-chip) of the nodes is transparent to the programmer. Figure 2 depicts this concept. Two CAN lines (red and blue) connect an arbitrary number of on-chip and off-chip nodes. On the boundary between on-chip and off-chip sits a VCAN gateway that implements a physical CAN connectivity. Each node deploys a VCAN Provider which implements the CAN API.

A. CAN System Model

A CAN system [9] is a multi master network which uses a CSMA/CD (Carrier Sense Multiple Access/Collision Detection) variant with arbitration on message priority. Each message is assigned a fixed priority/ID and a sequence of messages forms a signal. Before sending a message the CAN controller checks, if the bus is busy, then starts transmission. If two nodes are simultaneously starting to send, the one with the higher priority will win the arbitration and the other will back-off.

In order to support a wide variety of legacy CAN applications the VCAN gateway service supports the BasicCAN (receive queues) and the FullCAN (mailboxes) behavior. BasicCAN and FullCAN are not defined in the specification of the CAN standard, but describe two different principles in the architecture of a physical CAN controller, considering mainly the way how incoming messages are filtered. BasicCAN is usually used in cheaper standalone CAN controllers or in smaller microcontrollers with an integrated CAN controller. A BasicCAN controller has a single FIFO receive-queue with a global acceptance filter. FullCAN is used in high performance CAN controllers and microcontrollers. FullCAN controllers have a set of buffers called mailboxes that are assigned an identifier and are set to work as transmit, receive or remote buffers. When the CAN controller receives a data message it checks the mailboxes in order to see whether there is a mailbox configured as a receive buffer that has the same identifier as the incoming message. If such a mailbox exists, the message is stored in the mailbox and the host is notified. Otherwise the message is discarded. When a remote message is received, the controller checks the message identifier against the mailboxes configured as remote buffers. If a match is found, the controller automatically sends a message with the identifier and data contained in that mailbox.

B. CAN Hardware

Physically, the VCAN Gateway consists of some hardware logic and resources (i.e., TISS, NIOS2 processor, some on-chip memory, local IO, pins) on the FPGA and an expansion board. The expansion board uses an M51–Quadruple CAN Interface, which provides four distinct SJA1000 CAN controllers[5]. Thus, we support with this design up to four different physical CAN lines. Each CAN controller can be operated in one of two modes: BasicCAN mode or PeliCAN mode. For the VCAN overlay only PeliCAN mode is used, because this mode also covers the BasicCAN mode. Moreover, the communication speed for each controller can be configured individually from 62.5 kbits/s up to 1 Mbits/s.

For outgoing messages each CAN controller provides a transmit buffer that is able to hold one message. This message is autonomously transmitted after a successful bus arbitration. A receive buffer with 64 bytes allows to receive between 4 and 21 messages, depending on the frame format and number of data bytes per message. When the receive buffer is full, messages have to be read from the buffer before further messages can be received. Otherwise, messages can get lost. In order to guarantee the same message order for components inside the MPSoC as well as receivers outside, the CAN controllers are configured to also receive messages that have been sent by the CAN controller itself. Thus, virtual, on-chip CAN messages are not treated differently to messages on the

[4]Note that the CAN overlay is fully functional without AUTOSAR. We point out this connection only because our case will make use of this potential integration.

[5]The actual number of transceivers is arbitrary and not constrained by our design. Four simply seemed to satisfy our demonstrator's requirements.

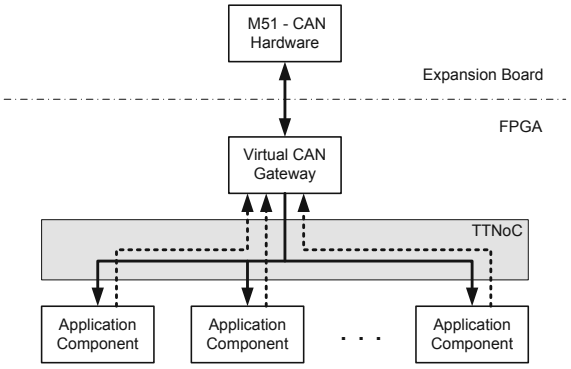


Fig. 3. On-chip and off-chip parts of the VCAN overlay

off-chip, physical CAN hardware. Only messages that won the bus arbitration and that have successfully been transmitted on the MPSoC external CAN network can be received by components inside the MPSoC.

The CAN hardware is connected to a gateway (marked as *IO Component* in Figure 3) via a memory-mapped interface that allows accessing the hardware by simply reading or writing memory locations. On this interface data has to be transferred in bytes. Each component within the MPSoC that uses the VCAN overlay is connected to the gateway through a periodic, unidirectional communication channel. Therefore, each component implements one single-cast channel to the gateway and the gateway implements a multi-cast channel to all registered nodes.

C. Virtual CAN Gateway

The VCAN Gateway acts as an intermediary between the endpoints of a CAN line. Endpoints can be either VCAN Providers (see Section III-D) on the application components, or CAN nodes connected to the off-chip network. The purpose of the VCAN Gateway is to implement virtual bus arbitration between CAN messages in order to privilege messages with a lower ID (i.e., higher priority). Furthermore, incoming messages have to be distributed to all components that are interested in a specific CAN line. As at the gateway implements no message filtering, all messages are forwarded to the VCAN Providers.

The VCAN Gateway Application consists of two tasks, one task is responsible for reading and writing to the CAN hardware, and the other task manages the communication with the VCAN Providers connected via the TTNoC. In order to synchronize the event-triggered hardware access task and the time-triggered virtual CAN communication task, both tasks implement receive queues and send buffers. Figure 4 depicts the structure of the VCAN Gateway.

A receive queue is an array of messages for an individual CAN channel that stores incoming messages until the time-triggered process collects them and sends them via the TTNoC. Send buffers hold one message per CAN line for each component. After a message has been sent successfully, the according send buffer is freed and the component is informed in order to allow further messages to be sent. Due to the semantics of

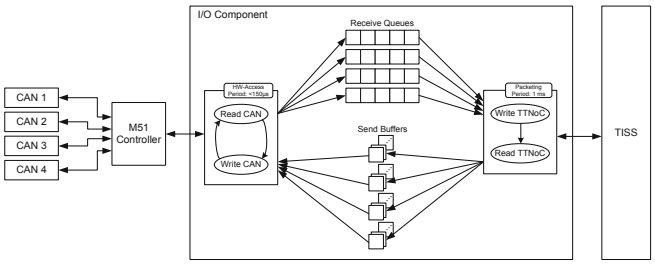


Fig. 4. VCAN gateway structure

CAN messages the following update strategies for send buffers are possible:

Send buffer empty: a new message can be placed in the send buffer without restriction

Send buffer occupied: if new message has

- *Higher message ID (i.e., lower priority):* new message cannot be copied to the send buffer
- *Same message ID:* new message replaces old
- *Lower message ID:* new message replaces old

1) Hardware access task: The hardware access task is responsible to write messages into the transmit buffers of the individual CAN controllers and to read messages from each of the receive buffers of the CAN controllers. Therefore, it checks each CAN controller whether the transmit buffer is ready for a new message to be sent. If yes, it searches the highest priority message (i.e., the message with the lowest message ID) from the send buffers that has to be sent on the according CAN channel. The send buffers contain at most one message per CAN line per component.

After the transmit buffer of all CAN controllers have been written, the task checks, if the receive buffer of the CAN controller contains messages. In case new messages are available, the controller copies them to the receive queues of the VCAN Gateway and the hardware buffers of the CAN controller are freed.

2) Periodic communication task: The periodic communication task manages the TTNoC communication with the VCAN Providers at the application components. It is activated with a frequency of 1kHz – this frequency is necessary to attain the requirements for the automotive case study (see Section IV). The task's activation instant is synchronized to the send and receive instants of virtual CAN messages by means of a task-trigger event emitted by the TISS.

At first, CAN messages are read from the receive queues and appended to the VCAN message sent on the TTNoC. This message has a static size and is designed to guarantee that no message will be lost at a CAN transfer rate of 1 Mbits/s. It additionally contains acknowledgment information about successfully sent messages for each individual component (for details on the virtual CAN message format see Section III-E).

After the virtual CAN message has been written to the TISS, incoming virtual CAN messages from the application components are read. These virtual CAN messages contain at most one CAN message per CAN line. The content of the CAN message is stored in the according send buffer. From

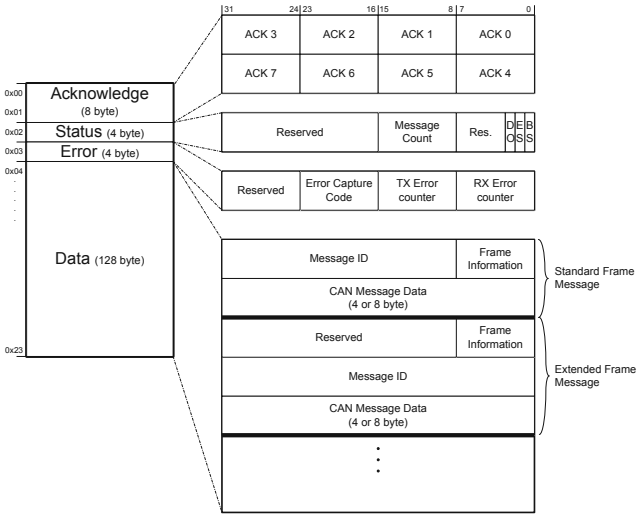


Fig. 5. VCAN message segment

there, the hardware access task constantly selects the message with the highest priority to be sent. Additionally, the virtual CAN message contains abort request information which tells the VCAN Gateway to cancel the transmission of a specific CAN message.

D. Virtual CAN Provider

The VCAN Provider is middleware running on an application component on top of a PikeOS instance. It enables applications to use the VCAN overlay. The VCAN Provider creates a virtual CAN controller locally. The Virtual CAN Provider is implemented as a file provider in the PikeOS and uses PikeOS' system extensions mechanism. Each VCAN line corresponds to one file provider. Each file provider contains all necessary information for the configuration of a CAN line, and it also provides memory containers for storing incoming and outgoing CAN messages. The VCAN Provider communicates with the TISS system extension on one side and the application software on the other side. The VCAN Provider is connected with the VCAN Gateway over two sporadic TTNoC ports.

E. Virtual CAN Message Format

VCAN messages are used to distribute CAN messages, status information regarding the hardware, control information, and acknowledgment data about sent CAN messages between VCAN Providers and VCAN Gateway. There is one message type for each direction (corresponding to solid and dotted lines in Figure 3). Both types of VCAN messages reserve bandwidth for each of the four CAN channels. Thus, VCAN messages consist of four segments having the same structure.

The benefits of this setup is that (a) no CAN line is influenced by the traffic load of another CAN channel and no message gets lost even in full load scenarios, because of no shared resources, and (b) it enables direct access to the required CAN segment, since the address offset within the virtual CAN message is known.

1) Virtual CAN Gateway message: The Virtual CAN Gateway message is a single message that is transmitted to all VCAN Providers simultaneously. Figure 5 depicts the structure of one segment of a VCAN message. Each segment starts with an acknowledgment block that contains an **ACK** field for each component. This field is an 8 bit sequence number, which indicates to the VCAN Provider that a CAN message has been sent successfully on the CAN line. If the *ACK* field is 0, then no message from the according VCAN Provider has been sent on the CAN line in the last period. Otherwise, the sequence number that has been sent by the Virtual CAN Provider in the transmit request message is returned. The next element in the message structure is the **status word** which is divided into Bus Status (BS), Error Status (ES), Data Overrun (DO), and Message Count record fields.

The BS flag is read from the CAN controller and signals whether the controller is involved in bus activities, or not. When the **ES** flag is set, an error occurred, which cause can be read from the error capture code. Is the **DO** flag set it means, that an incoming CAN message has been discarded as the receive buffer was full. **Message count** contains the number of CAN messages that are transmitted in the Data section of the virtual CAN message.

In the **error status word** a copy of the content of different hardware registers of the CAN controller is provided. This includes the **RX and TX Error counters**, as well as the **Error Capture Code**, that may indicate the cause of a communication or controller error.

The **data section** of the VCAN Gateway message is reserved for the actual CAN messages that have been received by the CAN controller. The size of this section (i.e., 128 byte) is designed to guarantee no message loss at a CAN communication speed of 1 Mbits/s and a virtual CAN message period of 1 ms. In order to save bandwidth, this section is organized as one continuous data block where a CAN message only consumes as many data words as needed. In this data section, CAN messages are further distinguished according to their frame format. For standard frame messages the Frame Information and the Message ID fields share one 32 bit data word. Extended frame messages use separate data words for the frame information and the message ID. Depending on the number of bytes in the CAN message data section, one (i.e., for $1-4$ data bytes) or two (i.e., for $5-8$ data bytes) data words in the virtual CAN message have to be used. Thus, for each CAN message between 1 and 4 data words with 32 bit are consumed.

2) Virtual CAN Provider message: The Virtual CAN Provider message is an individual message from each of the Virtual CAN Providers to the Virtual CAN Gateway Application. It is used to communicate transmit requests, which require that a CAN message that has to be sent is added, or abort requests (or both). In Figure 6 the structure of one segment of the VCAN Provider message is shown. Beside the CAN message content (i.e., Frame Information, Message ID and Data) this message type contains a **Transmit Request Sequence Number** and an **Abort Request Sequence Number**. Only if the sequence number is not equal to zero, the according command is valid.

The transmit request sequence number is incremented

whenever a new CAN message has to be sent. If no CAN message is available, it is reset to 0. This number will be returned by the VCAN Gateway when the CAN message has been sent successfully on the CAN line. When the transmission of a CAN message should be aborted, the abort request sequence number is set to the sequence number of the transmit request to be deleted. If the CAN message has not been sent, the VCAN Gateway will delete the referenced CAN message from the send buffers and CAN controller registers.

In contrast to VCAN Gateway messages, CAN messages that are contained in transmit requests always have the same structure and reserved message size.

F. Virtual CAN Protocol

The VCAN protocol defines the sequence of messages exchanged on the TTNoC and the off-chip CAN network. For incoming CAN messages (i.e., originating from an off-chip node), this sequence is straightforward:

1) M51 CAN controller receives CAN message
2) VCAN Gateway reads CAN message from controller
3) VCAN Gateway appends CAN message to VCAN Gateway message
4) VCAN Provider receives and processes the VCAN message

The sequence for outgoing CAN messages (i.e., originating from some application component) is as follows:

1) VCAN Provider sends CAN message with Transmit Request Sequence Number (TRSN) to VCAN Gateway
2) VCAN Gateway tries to send the CAN message on the CAN network
3) If CAN message can be transmitted on the CAN network, the message is simultaneously read by the CAN controller
4) CAN message is read back (i.e., read from the receive buffer of the CAN controller) by the VCAN Gateway and treated like a normal incoming CAN message
5) CAN message is packed into the VCAN Gateway message and the TRSN is copied into the ACK field of the application component from which the CAN message originates.
6) VCAN Provider reads the VCAN Gateway Message and uses the acknowledgment information to free the send buffer

The VCAN Provider has to wait at least one complete period until it receives the acknowledgment for a transmit request. But due to the send buffer update strategy (see Section 1.2), the VCAN Provider may send a new transmit request (with incremented TRSN) to the VCAN Gateway Application even if the old transmit request has not yet been acknowledged. This may only happen if the message ID of the new CAN message is lower or equal to the previously sent transmit request. CAN message with higher message ID can only be transmitted when the old transmit request was acknowledged. Because of the TRSN and the ACK field, the VCAN Provider always knows which of the transmit requests is acknowledged.

In Figure 6 an exemplary protocol sequence is shown. The VCAN Provider sends a transmit request ($TR = 27$) for a CAN message with $ID = 48$. As no CAN messages have been receive on the CAN network until the first VCAN Gateway Message is generated, this message does not contain any CAN message and an empty ACK field (i.e., $ACK = 0$). After an (unrelated) incoming CAN message with $ID = 35$ was received, the VCAN Gateway could send the CAN message from the transmit request 27 on the CAN network. Simultaneously, the CAN message is read by the CAN controller.

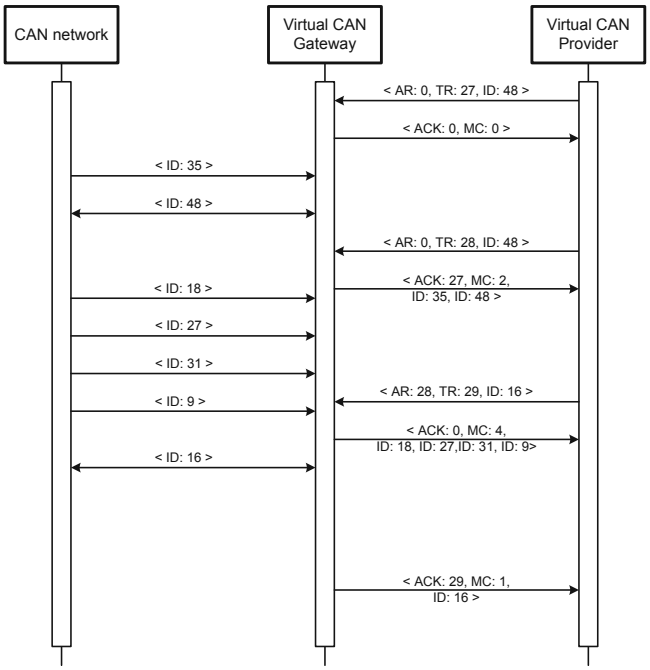


Fig. 6. Exemplary VCAN protocol execution

The VCAN Provider sends another transmit request ($TR = 28$), even if the old request was not yet acknowledged. Thereafter, the VCAN Gateway transmits the CAN messages that have been received since the last VCAN Gateway Message (i.e., messages 35 and 48). As the CAN message of transmit request 27 has been successfully sent, the ACK field returns the sequence number of this request ($ACK = 27$).

During the next period the VCAN Gateway is not able to send its CAN message, as all incoming messages have higher priority on the CAN network. When the VCAN Provider sends its next transmit request ($TR = 29$), the message also includes an abort request command ($AR = 28$). This tells the VCAN Gateway to remove the CAN message from the previous request from the send buffers. The next VCAN Gateway Message does not acknowledge any transmit request, but contains all CAN messages that have been received in the last period.

Finally, the VCAN Gateway send the CAN message from the last transmit request on the CAN network and sends the acknowledgment at the end of the period.

IV. Case Study: Hybrid Vehicle Control

The automotive case study demonstrate how the ACROSS MPSoC can be used to integrate several ECUs on a single chip.

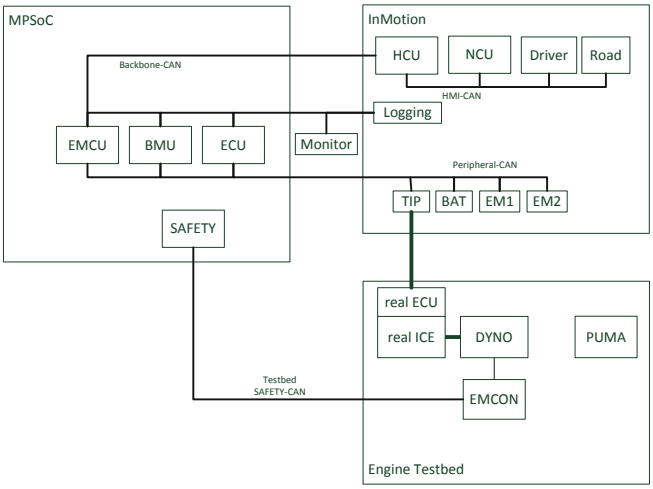


Fig. 7. System model of the case study

It shows how the VCAN overlay is used to connect the single ECUs and how it integrates with a sophisticated automotive tool chain. The goal of the used hybrid control application is to minimize fuel consumption.

The case study realizes a HiL setup. Figure 7 and Table I depict and describe the according system model. A HiL setup typically consists of a unit under test (the MPSoC), an environmental simulation and optionally a physical testbed. We used the AVL InMotion[6] HiL tool which offers an interface to different kinds of automotive testbeds. In the automotive testbed we used a custom hybrid engine.

For validation of the correct functioning of the VCAN infrastructure, we did some integration tests. Next, we considered the communication between InMotion and the MPSoC in the hybrid engine control scenario. This communication is done using two CAN lines (Backbone-CAN and Peripheral-CAN). Measurements were made with the AVL InMotion Software CAN-Analyzer.

A. Integration Tests

1) Message Completeness and Frequency Test:

Rationale Despite a high bus utilization, the completeness of messages and their respective frequencies must be guaranteed.

Procedure Both the MPSoC and AVL InMotion are connected to a CAN-Analyzer. The messages received on the CAN-Analyzer are checked against their reference values while frequency is increased. This is repeated for all CAN lines.

Result This test has been completed successfully.

2) Signal Calibration Test:

Rationale Signals (e.g., a speed signal) from the application must be packed into an according sequence of CAN messages by the sender and unpacked by the receiver.

Procedure Both the MPSoC and AVL InMotion are connected to a CAN-Analyzer. Each signal is brought to certain defined values (Using InMotion and the CAN-Analyzer). By checking the received signal for differences, this test indicates systematic and sporadic errors in definitions and implementation.

Result Sent and received signals were consistent.

3) Delay/Jitter – Tests:

Rationale Communication delays – or dead times – are of special interest in the area of control theory since they have a crucial impact on the control performance and even stability.

Procedure A CAN-Analyzer is used to introduce specially shaped (rectangle) signals on the CAN line, which are mirrored by different nodes and then read back by the CAN-Analyzer. By comparing the sent signal and the received signal, the transportation delay can be evaluated.

Result Delay times were nearly constant in all test cases. Delay values scatter around $500\mu s$. Thus, the VCAN overlay does not introduce significant delay or jitter.

[6]https://www.avl.com/

TABLE I. DESCRIPTION OF THE ELEMENTS OF THE CASE STUDY

Component	Description
HCU	The HCU (Hybrid Control Unit) implements the supervisory hybrid control strategy. It interfaces the Driver and commands EMCU and ECU.
NCU	The NCU (Navigation Control Unit) processes position information and estimates future loads based on the desired route.
Driver	The driver is a simulation that keeps the virtual vehicle on the virtual track in a desired manner.
Road	The road simulates external conditions and interacts with the virtual vehicle as well as with the Driver and the NCU.
EMCU	The EMCU (EMotor Control Unit) receives demands from the HCU and controls EM1 and EM2 considering constraints such as StateOfCharge.
BMU	The BMU (Battery Management System) monitors the vehicle's battery and calculates the StateOfCharge as well as other vital parameters.
ECU	The ECU (Engine Control Unit) receives demands from the HCU and controls the ICE (Internal Combustion Engine) of the vehicle.
SAFETY	This module implements a safety concept according to a machinery directive. It monitors the doors at the testbed, speed of engines, ...
DYNO	The Dyno is a powerful electrical drive controlled by EMCON. It is used to establish the simulated rotational speed at the shaft of the engine.
EMCON	Hardware and software for the control, manual and automatic operation of the combustion engine and dynamometer on an engine test bed.
PUMA	PUMA supports its users to execute testing tasks at the testbed.
TIP	TIP is the interface layer between AVL InMotion and the testbed.
BAT	The BAT is traction battery of the HEV. It is simulated within AVL InMotion and interfaces both EMotors and the respective Control Units.
EM1	EM (EMotor and Generator) is used to accomplish hybrid electric functionalities that lead to reduction of consumption and emissions.
EM2	See EM1
Monitor	In order to keep track of the communication between the ACROSS MPSoC and AVL InMotion. External components (CAN-Analyzer) are used. Also, inside AVL InMotion, a Data-Logging unit is realized.
Logging	

B. AUTOSAR Software Development

The workflow to derive the control application for hybrid vehicles is depicted in Figure 8. It conforms to the AUTOSAR specification and involves several tools from different suppliers. The input is basically the System Model, that describes the whole application running on the MPSoC, and the applications, which are the ECUs and the safety unit.

C. Hybrid Engine Control Use Case

1) Backbone CAN: The Backbone CAN line is used to send demand-values from the supervisory hybrid control unit (HCU) to the respective plant control units (ECU, EMCU, BMU) and to report measured values from the hybrid powertrain back. It uses a CAN speed of $1Mbit$ and Standard CAN (11bit) identifiers. In total there are 17 CAN messages defined that require a bandwidth of $15.04kb/s$.

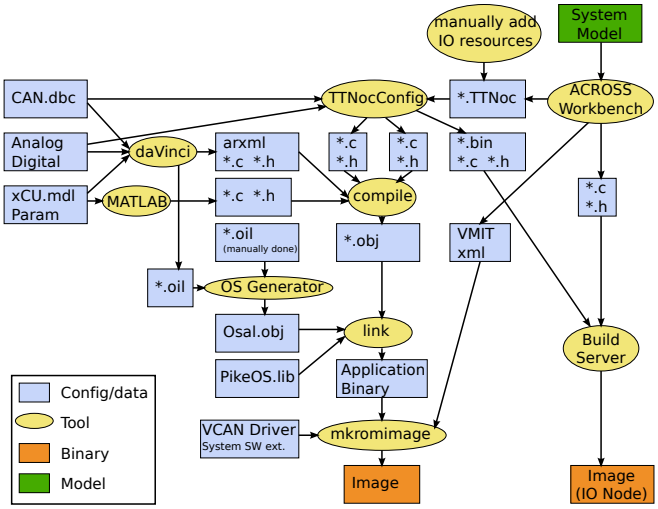


Fig. 8. Workflow and component view for automotive case study

2) Peripheral CAN: The Peripheral CAN line realizes the communication with powertrain-components such as ICE, EM and Battery and their according ECUs. In addition to this CAN line, several signals from sensors are fed into the system using analog and digital IO (and vice versa with actuator signals to powertrain components). In total there are 16 CAN messages defined that require a bandwidth of $13kb/s$.

A successful NEDC test [10] was conducted to assess the fuel economy of different hybrid control strategies.

V. Related Work

The CAN bus [9] was introduced by the Robert Bosch GmbH in 1986 and it became soon the de-facto standard for in-vehicle communication [11]. A multitude of works on the CAN bus is available. Many textbooks on real-time systems describe its working principle [12], [13]. Studies on its a capabilities and properties have been published, for instance, critical review on its real-time performance [14] or on its security [15].

The huge success of the CAN bus and the availability of low cost controllers extended its usage beyond the automotive industry and triggered new developments based on the original CAN protocol. Most notably, CANopen [16] and time-triggered CAN [17]. Also the applicability of the CAN bus as a serial communication protocol for the avionics has been considered [18]. Virtual CAN networks on top of a time-triggered network have been treated in [19]. A CAN gateway connecting physically distinct CAN lines is described in [20].

VI. Conclusion

In this work we presented a virtual network layer emulating a CAN bus. The resulting VCAN overlay facilitates the connection of purely virtual with physical CAN lines. Such a software layer is particularly useful when introducing new computation platforms like MPSoCs in existing production environments with established work flows and tool chains. We demonstrated our solution by implementing an automotive control application for hybrid vehicles on top of the VCAN overlay. The application was developed using the AUTOSAR design flow and tools.

Acknowledgment

The authors would like to thank Christian El-Salloum and Michael Kang for the fruitful cooperation. This research was in part supported by a Marie Curie IOF Action within the 7th Framework Programme under the funding ID PIOF-GA-2012-326604 (MODESEC). The responsibility for the content rests with the authors.

References

[1] M. Girardot and M. Schwarz, "Electronics-ization in automotive: Reinventing the industry model to boost profitable innovation," Cisco IBSG, Whitepaper, 2012.

[2] H. Wallentowitz, A. Freialdenhoven, and I. Olschewski, *Strategien in der Automobilindustrie: Technologietrends und Marktentwicklungen*. Teubner Verlag / GWV Fachverlage GmbH, 2009.

[3] M. Broy, I. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[4] R. Obermaisser, C. El-Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 7, pp. 956–965, 2009.

[5] C. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, "The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, 2012, pp. 105–113.

[6] C. Paukovits, "The Time-Triggered System-on-Chip Architecture," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Dec. 2008.

[7] R. Obermaisser and O. Hoftberger, "Fault containment in a reconfigurable multi-processor system-on-a-chip," in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, 2011, pp. 1561–1568.

[8] R. Kaiser and S. Wagner, "The pikeos concept: History and design," SYSGO AG, Whitepaper, 2007.

[9] *Road vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*, International Standardization Organization Std. ISO-11 898, 1993.

[10] T. J. Barlow, S. Latham, I. S. McCrae, and P. G. Boulter, "A reference book of driving cycles for use in the measurement of road vehicle emissions," TRL Limited, Published Project Report PPR354, June 2009.

[11] L.-B. Fredriksson, "Can for critical embedded automotive networks," *Micro, IEEE*, vol. 22, no. 4, pp. 28–35, 2002.

[12] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

[13] W. Voss, *A Comprehensible Guide to Controller Area Network*, 1st ed. Copperhill Media Corporation, 2005.

[14] H. Kopetz, "A comparison of can and ttp," in *Proc. 15th IFAC Workshop on Distributed Computer Control Systems (DCCS)*, 1998.

[15] R. Kammerer, B. Fromel, and A. Wasicek, "Enhancing security in can systems using a star coupling router," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, 2012, pp. 237–246.

[16] *Industrial Communication Subsystem based on ISO 11898 (CAN) for Controller-dDevice Interfaces - Part 4: CANopen*, CAN-in-Automation (CiA) Std. DS/EN 50 325-4.

[17] *Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication*, International Standardization Organization Std. ISO-11 898-4:2004.

[18] C. Lin and H. Yen, "Reliability and stability survey on can-based avionics network for small aircraft," *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 2, p. 8, 2005.

[19] R. Obermaisser, "Reuse of can-based legacy applications in time-triggered architectures," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 4, pp. 255–268, 2006.

[20] Infineon, "Multican – CAN-gateway functionality witout cpu interaction," Infineon Technologies AG, Application Note AP29005, 2007.

CHAPTER 3

Secure Channels in an Integrated MPSoC Architecture

Secure Channels in an Integrated MPSoC Architecture

Haris Isakovic, Armin Wasicek
Institute of Computer Engineering
Vienna University of Technology
Email: {haris.isakovic, armin.wasicek}@tuwien.ac.at

***Abstract*—Providing security in an embedded system often boils down to solving a trade-off problem between security and performance. Simultaneously, Multi-Processor System-on-a-Chip (MPSoC) devices are in the early stages to increase computational performance, energy and die area efficiency, and reduce the number of physical units in the embedded system design arena. Moreover, MPSoCs enable composing heterogeneous subsystems on a single silicon die which is particularly desirable for large volume embedded devices. However, these benefits come at a price: an increase in the system's complexity. Complexity does not only make the system design process more difficult, but also it renders certain vulnerabilities possible. A solution is to follow well-established architectural principles to reduce complexity and to provide the required level of security. In this paper we demonstrate how the basic architectural principles of the ACROSS MPSoC architecture can be combined with the requirements of standard security techniques (i.e., encryption, authentication) to produce an efficient security solution for MPSoC systems. We propose a security architecture which uses the principles of temporal and spatial partitioning, temporal determinism, and mixed-criticality integration to migrate resource expensive security functions form the application components to a dedicated security component within the MPSoC. This leaves application components with a thin security provider, without any loss of functionality and more local resources at their disposal. Thereby, we deliver a flexible, resource efficient security solution, which highlights the benefits of partitioning MPSoC architectures for security.**

I. Introduction

A security architecture describes how the security design artefacts are positioned and invoked, and how they relate to the overall system architecture. A security architecture's goal is to establish the non-functional properties confidentiality, integrity, and availability for authorised users [1]. Technically, the portion of the system that sustains these three properties is called the Trusted Computing Base (TCB). A TCB [2] is a *small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security*. Hence, defining what is inside and what is outside a system's TCB is the critical security design task.

Integrated systems implement a variety of different types of computational units. They are a very challenging category for system designers, because each subsystem can have completely different properties. Multi–Processor System–on–a–Chip (MPSoC) platforms are currently emerging as a very powerful representatives of heterogeneous systems. The flexibility of an MPSoC with heterogenous components can facilitate distributing the application's computational load between components. This can be a particular benefit in embedded system applications, which often require a special mix of processing tasks.

However, these benefits of heterogenous systems come at a price. According to the European Aviation Safety Agency (EASA) [3], most existing MPSoCs are classified as 'highly complex micro controllers'. With respect to security, complexity is a major source of vulnerabilities [4]. Therefore, a MPSoC architecture should provide complexity reduction techniques not only for system design purposes, but also to facilitate a more secure system [5]. The most common engineering approach to reduce complexity is *divide et impera* – partition the system in controllable subsystems. Then, the problem to be solved by the architecture is how to join the subsystems in a secure way.

Another challenge on the road to a secure system is the tradeoff between performance and security. Mechanisms used to enforce security are usually resource demanding. Particularly, in resource-constrained environment like embedded systems this tradeoff is a critical point for security design. In safety-critical systems, resource allocation is often guided by over-dimensioning in order to provide enough resources at a critical instant. This design approach might lead to an expensive design solution.

In this paper, we propose a possible security architecture based on the ACROSS MPSoC [6]. The ACROSS MPSoC facilitates a solution for the design and implementation of hard real-time applications. It represents a new generation of MPSoCs with integrated Time-triggered Network–on–a–Chip (TTNoC). The ACROSS MPSoC provides several non-functional properties (i.e., temporal determinism, temporal isolation, spatial isolation, partitioning). The main contribution of this paper is to show, how these properties can be used in combination with standard security mechanisms to improve the organisation and implementation of a generic security architecture.

The remainder of the paper is organized as follows: Section II focuses on partitioning architectures, Section III provides overview of security architecture basics, Section IV elaborates how partitioning is used to design security, in Section V we present our findings, and Section VI concludes the paper.

II. Partitioning Architectures

In this section we describe the concept of partitioning in computer systems. Our main focus is on partitioning in the

embedded systems domain and MPSoC platforms.

Kopetz [7] illuminates three major design strategies to counteract complexity for the embedded system designer: partitioning, abstraction, isolation and segmentation. Numerous examples of partitioning can be found in biology (i.e., ant colonies [8]), as well as in system design. Partitioning a highly complex computer systems into smaller independent functional units is one of the most frequent techniques to master complexity. A common definition of a partition is *'a system partition or logical partition is a subset of computer's hardware resources, virtualized as a separate computer'* [9]. This definition mostly refers to software partitions implemented through a virtualization, but there are other ways to implement partitioning on a computer system. If we observe partitions as independent functional units with certain properties, than each sub-system of a larger system which possesses these properties can be called logical or system partition. In practice, a partition or a functional unit is realized as a software/hardware artefact that works in isolation, but is connected via well-specified interfaces to the other parts of the system. Following this definition it is apparent that a partition's properties manifest in its interfaces. Thus, we can deduct all required and existent properties of a partition by analyzing its interfaces [10]. Clearly, the concept of a partition is very well applicable in the security domain, because a security architecture first partitions a system in different entities and then defines access rights between these entities.

Two properties of a partition are very relevant for the design of embedded real-time systems:

- *Spatial Isolation*. A spatially isolated partition owns a part of memory which cannot be accessed by any other partition. The guarantee of the spatial isolation increases confidentiality and integrity of the partition's private data. These two properties are also part of the security specification of a partition and the system in general.
- *Temporal Isolation*. A temporally isolated partition has a guaranteed uninterruptible execution time by any other partition. This means that temporal characteristics of that partition cannot be influenced in any way. This property increases execution determinism, it can be used to predict a system behaviour. Therefore it supports dependability and reliability of a partition, which are preconditioning properties for safety.

In order to apply the principle of partition on a subsystem, at least one of these properties must be achieved. Other important properties are: energy consumption, computational power, security, fault propagation. The partitions are independent units and if they are spatially and temporally isolated their additional properties mutually incremental. The properties of an individual partition add up to the same property on the system level.

In order to analyse a computer system, we want to apply the concept of partitioning on two levels:

- *Software Partitions*. The partitioning mechanisms are realized in software, for instance, in the operating system.
- *Hardware Partitions* are implemented by dividing a system's physical resources (i.e., processors, memories, communication controllers) into functionally independent units.

In the next two sections we describe the methods and techniques used in the state-of-the-art partitioned systems, to achieve partitioning. We show how these different types of the partitions stack up, in the systems with mixed partition types (i.e., MPSoC).

A. Operating System Level

On the Operating System (OS) level, techniques to achieving partitioning are often summarised under the hood of virtualization. Virtualization has been introduced by IBM in 1972 for the mainframe computers [11]. Since then virtualization became important for every computing domain, from powerful internet servers to small embedded systems. There is a large spectrum of relevant techniques, ranging from simple process virtualization to virtualizing an entire computer system.

A virtualization manager is the part of the system that spawns the partitions. It is usually called Hypervisor or Virtual Machine Monitor (VMM). A Hypervisor can be implemented as a process that emulates an OS. Therefore, it can contain again different processes which might translate to execution threads of the original OS hosting the Hypervisor. This technique is called Paravirtualization. In this case, the Hypervisor encapsulates an application in a process. Consequently, the strength of the isolation between encapsulated applications depends on how the original OS manages the resources between different processes. Other virtualization techniques are full virtualization [12] and Hardware Assist, which relies on special hardware modules which support the virtualization. The full virtualization uses binary translation technique and direct execution techniques. This means that Hypervisor takes the instruction set used by the Guest OS and translates it to the one executed on the underlying hardware.

Another way to achieve software partitioning are microkernels. Contrarily to virtualization whose origins are in the mainframe computers, microkernels were soon adopted by the embedded computing community. A microkernel provides only the most basic operations needed to implement an OS. Other functionalities (e.g., a networking stack) are shifted in so-called servers which run as processes in the user space. Microkernels are an important architecture type for Real-Time Operating Systems (RTOSs). In the embedded domain important representatives of the microkernel-based OSs are INTEGRITY [13], LynxOS [14], OKL4 [15] and PikeOS [16].

Another way to achieve software partitioning is using a microkernel. A microkernel provides only the most basic operations needed to implement an OS. Other functionalities (e.g., a networking stack) are shifted in so-called servers which run as processes in the user space. Microkernels are an important architecture type for RTOSs.

In the embedded domain some of the microkernel-based OSs which implement partitioning are INTEGRITY [13], LynxOS [14], OKL4 [15] and PikeOS [16]. T

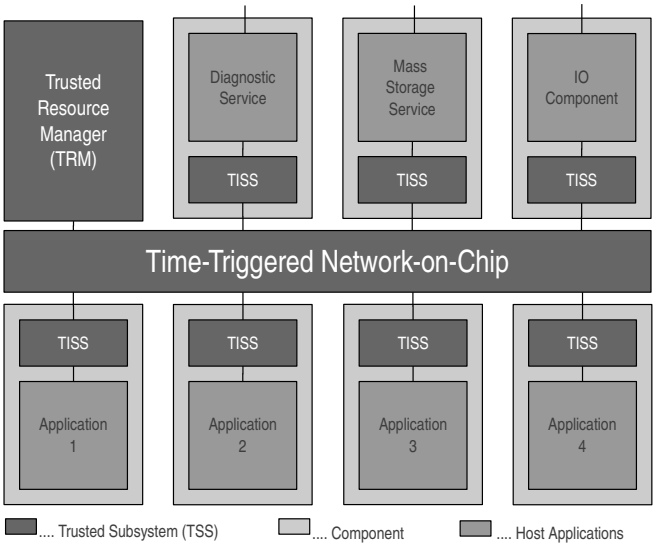


Fig. 1. Schematic diagram of ACROSS MPSoC: Application (1,2,3,4) and system components (Diagnostics, Mass Storage, IO) are composed via the Time-Triggered NoC

B. MPSoC Level

The future of embedded systems is in MPSoCs. Microcontrollers, Field–Programmable Gate Arrays (FPGAs) and other common computing elements will be pooled on a single silicon die shaping a heterogeneous System–on–a–Chip (SoC) for energy efficiency, size, cost, and integration reasons. These elements will be connected by a dedicated communication infrastructure like point-to-point (P2P) channels, a bus, or Network–on–a–Chips (NoCs) [17]. MPSoCs are a powerful solution, but apart from the benefits like versatility and performance there are some drawbacks like complexity. For instance, using an MPSoC in a safety-critical applications is currently not advisable, because the certification issues.

As a research initiative, the ACROSS MPSoC (see Figure 1) is set out to simplify certification for safety-critical applications in different application domains. This is achieved by implementing partitioning and encapsulation. Opposed to some virtualization techniques that use hardware features, the ACROSS MPSoC is totally designed to segregate its components in partitions in hardware. A component is a container for any system or application functionality. Communication between partitions is done through so-called encapsulated communication channels that are defined over a Time-Triggered NoC [18]. End-points of these encapsulated communication channels are called *ports*. The generic interface used for message exchange between SoC components and the TTNoC is called Trusted Interface Subsystem (TISS). Messages exchanged over the TTNoC are strictly bound to their temporal specification. Furthermore, memories are not shared, each partition is required to have its own. Components are containers for functionalities. Application components realize the required functions for the application whereas system components are responsible for correct operation of the MPSoC's native functions (i.e., IO, Mass Storage, Synchronization, etc).

At first, the ACROSS MPSoC was designed to host hard real-time systems. But the applied design principles have a strong impact on the implementation of a security architecture. Using the powerful concept of partitioning on the hardware level, we are able to imitate physically distinct units of computation. A similar design can be found in current industrial-grade SoC solutions like the ARM TrustZone (secure and non-secure mode of operation) [19] and the Cell processor's 'Secure Vaults' [20].

C. Combination

Hardware and software partitioning mechanisms can be combined to facilitate a two-level partitioning approach. In ACROSS, a component can instantiate a partitioning OS like for instance PikeOS. This results in a fine grained partitioning architecture. Each software partition spawned inherits the properties given through the execution on a specific hardware.

In our case, a PikeOS partition will execute on a component of the ACROSS MPSoC. Therefore, it has some very special properties when communicating with other partitions on the same MPSoC. Although integrated on the same SoC two instances of PikeOS cannot invalidate temporal or spatial guarantees of each other. From the application's view the usage of a partition is always the same. Partitions communicate through so-called queuing and sampling ports, be it that they are hosted on the same component or on different components. This gives a designer a the freedom to split the application in as many partitions as needed and distribute the partitions on components as wanted. When communicating with partitions on the same components, a partition has the usual properties that are given by PikeOS. When connecting to an external device, a partition acts as a physically distinct entity.

In the next section we present a generic security architecture for communication that we will put on top of the partitioning concept of the ACROSS MPSoC in the subsequent section.

III. Secure Communication Architecture

A security architecture is the part of the overall system architecture, which describes how does the system satisfy security requirements [21]. In this section we describe one of the most common design pattern of any security architecture, a *Secure Channel*. Associated with a Secure Channel are the Secure Kernel and a key management facility.

Figure 2 depicts the big picture of the secure communication architecture. It depicts the architectural elements that refine the TISS which is the standard interface to access the TTNoC. On the bottom, a regular communication through a standard channel without any security guarantees is given. If a Secure Channel should be used, the send and receive operations are executed on the Secure Provider which transparently provides a channel with security properties to the application. All Secure Channels of a component are administered and maintained by the Secure Kernel.

A. Secure Kernel

The Secure Kernel is considered to be the TCB in an ACROSS MPSoC. The security of a system cannot be compromised if the TCB is operating according to specification. The Secure Kernel is responsible to manage keys and access

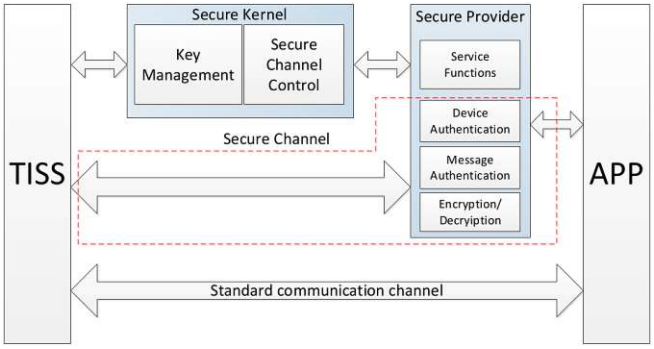


Fig. 2. Block diagram of Secure Communication Architecture

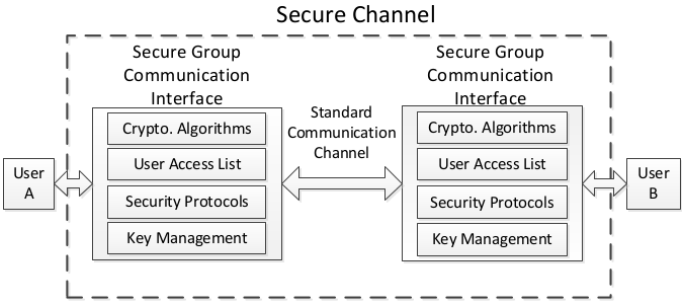


Fig. 3. Secure Channel Structure

rights of all Secure Channels of a component. Therefore, an application does not have to take care of any bookkeeping, but it can invoke the communication API for secure and regular channels in the same way.

The Secure Kernel is an independent thread which executes several tasks important for establishing of secure channels. It executes in its own partition and is therefore isolated from the application's functions. Its memory cannot be accessed from the application partitions and it embodies only the minimal amount of confidential information. Hence, even if it would be compromised, the effect would concern only the active connections of the compromised component. An attacker could not use it to gather keys from other components.

When the channel is opened from an application a key for that channel is requested from the Secure Kernel. This request contains the port ID of the channel, which is then used as an input for the minimal perfect hash function. The hash function returns the index of the corresponding key in the key table managed by the Secure Kernel. If keys have to be exchanged with some other entity, the Secure Kernel performs this action in a secure and reliable manner.

B. *Secure Channel*

The Secure Channel is an abstract object composed of a regular communication channel and a security protocol implemented on the communication channel. A protocol is defined as a series of steps, which involve two or more parties with a goal of achieving certain task. The security protocol uses cryptographic algorithms to solve a security related issue on a communication channel. Which security mechanisms are actually used in a security protocol is defined by requirements of an application. This can be implemented differently from application to application.

The basic structure of a secure channel is shown in Figure 3 and comprises following elements:

- Cryptographic algorithms which provide basic cryptographic services
- Security protocols which extend the capabilities of the cryptographic algorithms and provide security properties to communication channels
- A user access list to manage access rights to the channels
- Key management system provides secure storage and distribution of the keys used for various security protocols and cryptographic algorithms.

Establishing a Secure Channel requires coordination and connection of several basic services of the ACROSS service model. The secure channels in ACROSS are unidirectional single-cast channels based on symmetric encryption security mechanisms. Here the procedure that instantiates a Secure Channel in the ACROSS MPSoC:

1) Secure Channels are created by the application using the related API
2) The API evokes the Secure Channel Provider which opens a special storage holding unit for the channel. Thus, Secure Channels are host by a Provider and the Secure Kernel handles security key management.
3) Secure Channel Provider opens a single port on the TTNoC for each channel, which is then used to transmit data for the channel. The port configuration data is static and shared among channel among parties before compile.
4) A secure channel are used only for the off-chip communication, therefore a gateway (IO component) is instructed to establish communication with the other device. This requires configuring ports on both sides and the gateway to relay the messages.

C. *Key Management*

The key management defines the process of a key generation, key storage and a key distribution. Keys are generated either during the pre-configuration phase or during runtime. Each channel is provided with the single key which are managed by the Secure Kernels. The Key management exists once per physical device. Hence, it is implemented as a dedicated system component. Access is facilitated only via the rigorously specified interface at the TTNoC.

IV. Using Partitions to Design for Security

The security design problem we are solving in this section is how to map the general security architecture defined in the previous section to partitions. In our system model, partitions are established through the physical separation between cores as well as the separation mechanisms of the OS running on each core. Generally spoken, it is assumed that hardware security mechanisms are stronger than software mechanisms, because hardware is naturally not as modifiable as software.

Basically, we reason about partitions that execute on the same core and partitions that run on different cores. Partitions interact only via their communication (linking) interfaces. Communication channels are statically allocated before runtime. The integrity of the MPSoC's configuration is continuously checked by the Trusted Resource Manager (TRM). Then an encapsulated communication channel can be:

- *Local*: the parties are located on the same component and communicate via an OS mechanism like a message queue
- *On-chip*: the parties reside on different cores of the same MPSoC and they are communicating only using the TTNoC
- *Off-chip*: the parties reside on different MPSoCs and for the communication they use first the TTNoC, then a gateway to an off-chip network (e.g., TTEthernet)

A. Fine-grained control of encryption

Depending on the mode entities communicate Secure Channels are instantiated and the strength of their security mechanisms is determined. Embedded systems often do not dispose over bountiful resources and must compensate on some properties to achieve optimal performance. Each of the above described modes has a different impact on the realization of a Secure Channel. The concrete impact is determined by the attacker model used. For instance, we want to assume that an attacker can tap, eavesdrop, and manipulate the device's pins and their physical connections. It is out of scope for the attacker to open the device's package and read out the TTNoC[1] Furthermore, we also assume that reading out a component's memories (e.g., via microprobing) is not feasible.

Using such an attack model, we can determine which Secure Channels actually requires security processing and which considered to be safe. This has to be determined for each Secure Channel individually. Therefore, our architecture also accounts for instantiations with channels with mixed security requirements (i.e.,). Giving the designer such a fine-grained control over the execution of security protocols and hence cryptographic algorithms facilitates the design principle *Economy of Mechanism*. Moreover, the implementation of the security architecture can be more performant and less resource intense. This is particularly important for Embedded Systems, because they have often stringent resource constraints regarding computational power, memory sizes, and energy budget.

B. Offloading intense computations

The second direct benefit of our proposed generic security architecture is that computationally intense operations can be securely offloaded to system components. For instance, the key management which runs as system component in a separate hardware partition performs the creation of new cryptographic keys. Key creation can be a highly demanding process. At least, it involves computing a random number generator and conversion of random bits in the appropriate key format. In case of asymmetric encryption, primality testing and operations on big integer numbers are required in addition.

[1]If instantiated as an ASiC, it is easily to visually locate the connections of the NoC in the layout. It would have as well have a larger feature size than e.g., the devices implementing a processor.

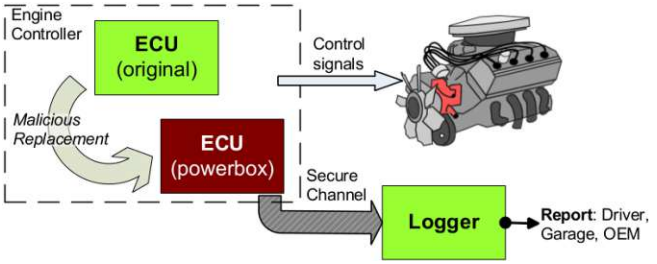


Fig. 4. Powerboxing: The original ECU is replaced by a powerbox that contains different (more powerful) control algorithms for a specific engine. In our experimental setup, a logger is connected via a Secure Channel to detect such a manipulation.

The key management has to solve a producer-consumer problem. It has to produce enough cryptographic keys that are then consumed by the respective Secure Kernels. Traditionally, key creation is implemented as a background task on the same computer that perform send/receive operations. This has a major drawbacks: considering a real-time system, this background task might not receive enough resources in a critical instant and the application could run out of cryptographic material. When integrating several subsystems into a single device like aspired by the ACROSS MPSoC, the benefits of a shared key management component is getting obvious: The components can focus on what they actually want to do, namely sending and receiving messages. Key creation can be managed by a specialised component. This has also an impact on the energy consumption of the device. If no background tasks are running, power gating is trivial. Moreover, if the key management component has a dedicated hardware support, its operations can be perform energy-efficiently.

V. Use Case: Power Boxing

As a use case, we use a Secure Channel to monitor values of an automotive Engine Control Unit (ECU) to detect malicious manipulations in the control system software, for instance through a power boxing attack [22].

Powerboxing [23] directly modifies the output signals of an ECU by inserting a hardware module in the vehicle (see Figure 4). The rationale behind this idea is similar to chip tuning: an attacker wants to tap the engine's maximum potential. The inserted module either replaces the original ECU on the communication system or it is placed as a man-in-the-middle between the original ECU and its connected actuators. The installation in the vehicle is fairly simple as demonstrated by several supplier guidelines and demonstration videos on the internet. It requires plugging the new unit to the power supply and to reconnect the network cables from the original ECU to the powerboxing module. Example is a fuel injector tuning chip which changes signals between ECU and a fuel injector module. It increases power of the engine by increasing the injection time of the injector. The principal block diagram is shown in Figure 4.

Such manipulation can be effectively detected by continuously monitoring an ECU's output. We used a Secure Channel

to periodically output an authenticated message containing the current state of the engine controller. The controller is implemented in an ACROSS MPSoC. The receiving device logs the state and its authenticity. Absence of the correct authentication indicates external manipulation of the signals. The logger might then report to the driver, a garage, or even the Original Equipment Manufacturer (OEM) to inform that the current setup has been compromised.

The experimental setup for the original ECU was done by our colleagues from the control department partly in simulation and partly with real devices. In a subsequent step, we added the Secure Channel to the demonstrator as a proof-of-concept for our security architecture. Moreover, we showed that the operation of the Secure Channel does not interfere with the temporal guarantees given by the component, hence, it is minimally invasive.

VI. Conclusion

Partitioning is a valuable concept to effectively reduce complexity in system design. In this paper, we showed how partitioning can successfully applied to design and implement a secure communication architecture. Starting with the abstract notion of a secure channel we broke down the required security protocols and cryptographic algorithms in different partitions where the respective functionalities are implemented. This straight-forward design approach is enabled by the strong isolation and encapsulation guarantees of the underlying system architecture. In our case be build on an MPSoC architecture intended for integrated, hard real-time systems. Layering the presented security architecture on top does not interfere with established timeliness and dependability properties as we demonstrated in the automotive case study. Wrapping up, we showed that implementing security is feasible even in strictly constrained environments like hard real-time systems, if a sound architectural approach is followed. Clean separation of functionalities is enabled by implementing partitioning as a foundation.

Acknowledgment

This document is based on the ACROSS project in the framework of the ARTEMIS programme. The work has been funded in part by the ARTEMIS JU and National Funding Agencies of Austria, Germany, Italy and France under the funding ID ARTEMIS-2009-1-100208. This research was in part supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme under the funding ID PIOF-GA-2012-326604. The responsibility for the content rests with the authors.

References

[1] O. S. Architecture. It security architecture. [Online]. Available: http://www.opensecurityarchitecture.org/cms/en/definitions/it-security-architecture

[2] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 265–310, 1992.

[3] E. A. S. Agency. Development assurance of airborne electronic hardwa re. [Online]. Available: http://www.easa.europa.eu

[4] G. McGraw, *Software Security*. Addison Wesley, 2006.

[5] A. Wasicek and C. E. Salloum, "A system-on-a-chip platform for mixed-criticality applications," in *Proceedings of 13th IEEE International Symposium on Object/component/service-oriented Real-time distributed computing (ISORC)*, May. 2010.

[6] C. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, "The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, Sept., pp. 105–113.

[7] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[8] M. Albrecht and N. Gotelli, "Spatial and temporal niche partitioning in grassland ants," *Oecologia*, vol. 126, pp. 134–141, 2001. [Online]. Available: http://dx.doi.org/10.1007/s004420000494

[9] K. Singh. Security on the ibm mainframe. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247803.pdf

[10] H. Kopetz and N. Suri, "Compositional design of rt systems: A conceptual basis for specification of linking interfaces," in *Research report, Technische Universitt Wien, Institut fuer Technische Informatik, Treitlstr. 1-3/182-1, 1040*, 2003, pp. 51–60.

[11] I. Corp. z/vm^{TM} built on ibm virtualization technology. [Online]. Available: http://www.vm.ibm.com/pubs/HCSF8A50.PDF

[12] vmware. Understanding full virtualization, paravirtualization, and hardware assist. [Online]. Available: http://www.vmware.com

[13] G. H. Software. Integrity rtos. [Online]. Available: http://www.ghs.com/products/rtos/integrity.html

[14] LynuxWorks. Lynxos-178 rtos - lynuxworks. [Online]. Available: http://www.lynuxworks.com/rtos/rtos-178.php

[15] O. K. Labs. Okl4 microvisor. [Online]. Available: www.ok-labs.com/products/okl4-microvisor

[16] S. AG. Pikeos rtos and virtualizations. [Online]. Available: http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/

[17] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 23:1–23:20, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1255456.1255460

[18] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *RTCSA*, 2008, pp. 120–129.

[19] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," *ARM white paper*, 2004.

[20] K. Shimizu, H. P. Hofstee, and J. Liberty, "Cell broadband engine processor vault security architecture," *IBM Journal of Research and Development*, vol. 51, no. 5, pp. 521–528, 2007.

[21] M. Gasser, *Building a secure computer system*. New York, NY, USA: Van Nostrand Reinhold Co., 1988.

[22] A. Wasicek, "Copy protection for automotive electronic control units using authenticity heartbeat signals," in *10th IEEE International Conference on Industrial Informatics (INDIN)*, 2012, pp. 821–826.

[23] H. Prankl and H. Schaufler, *Motortuning zur Leistungssteigerung an Traktoren: Bericht ; Projekt-Nr. BLT 05 3322*. FJ BLT, 2006. [Online]. Available: http://books.google.at/books?id=fto-MgAACAAJ

CHAPTER 4

A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform

A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform

Haris Isakovic
Institute of Computer Engineering
Vienna University of Technology
Email: haris@vmars.tuwien.ac.at

Radu Grosu
Institute of Computer Engineering
Vienna University of Technology
Email: radu.grosu@tuwien.ac.at

***Abstract*—There is a huge discrepancy between off-the-shelf (COTS) hardware architectures and requirements for embedded industrial applications. Industrial systems are getting more complex by the day, and an interaction of highly diverse components within these systems is unavoidable. An implementation of such systems on COTS hardware is challenging. Platforms based on single-core CPUs is becoming limited, and use of multi-core architectures yields safety risks, and overall inefficiency. Tailored architectures provide adequate service but they lack flexibility and therefore their economic justification is limited. Emerging technologies i.e., hybrid system-on-chip combined with novel architectural concepts are filling blind spots between COTS architectures and embedded industrial applications.The paper presents the implementation of an MPSoC architecture on a hybrid system-on-a-chip platform. This architecture provides unique capabilities for embedded applications, in particular, the possibility to host mixed-criticiality and cross-domain applications.**

I. Introduction

Industrial embedded applications (such as, automotive, aerospace, railway, internet-of-things, industrial automation) are implemented either on general purpose COTS hardware or on dedicated hardware components. In the former case versatility and accessibility are main goals. In the second case the focus is on solving of a specific problem.However, the design and production of computer hardware architectures is mainly influenced by requirements of general purpose computing. The first computer architectures were based on single-core CPUs, which evolved to multi-core CPUs as the performance and efficiency limits of single-core were reached. Due to limitations of safety and reliability requirements the industrial applications are mainly implemented on single-core architectures. On a single-core CPU tasks are usually divided in time and the system software guarantees interference free operation. However, most multi-core processors are designed with no special consideration towards isolation of different tasks (e.g., all applications share components in non-deterministic fashion). As a consequence, a sufficient guarantees cannot be provided for safety critical systems without significant loss of performance, for example, disabling all cores, except for one. The industrial applications are getting more complex by the day and are forced to migrate multi-core architectures in order to meet the performance requirements, and the fact that CPU manufacturers are abandoning single-core architectures. The whole process is yielding serious challenges.

In this paper an overlaying hardware architecture is presented which utilizes advances in the design of state-of-the-art field programmable gate array (FPGA) devices technology. It is an adaptation of the hardware architecture described in [1], which proposes an alternative approach to COTS multi-core architectures. The legacy architecture was implemented on a platform with limited performance capabilities. In this paper a new implementation of the architecture is described, implemented on a novel hardware platform Arria V ST SoC from Altera [2]. It is implemented on a SoC chip that combines a powerful FPGA device with a hard processor (e.g., ARM Cortex). It elevates the performance capabilities of the overlaying architecture and provides new options for the system designers. The new implementation of the architecture uses a modular interface-based approach to build a heterogeneous MPSoC with hard processor component and a set of soft-coded processor components.

The following section gives an overview of the design challenges for the industrial embedded systems. In Section III the implementation of the legacy architecture is described. Further, Section IV introduces the hybrid SoC technology. Section V contains the implementation details of the architecture. In Section VI a short case study on two industrial uses cases is given, followed by sections with related work and future work. Final sections contain closing words and acknowledgments.

II. Challenges in Industrial Embedded Systems

As an answer to the growing complexity issues of computer systems, hardware vendors introduced a multi-processor system-on-a-chip (MPSoC) architectures. The goal is to integrate multiple systems on a single chip with the possibilities to share its infrastructure without interference, and to save on physical space and energy consumption. Embedded industrial applications are currently facing a number of imminent challenges and the MPSoC technology provides an alternative solutions to these problems:

Performance. A major problem for industrial embedded applications, especially for the applications with safety and real-time requirements, is how to replace single-core CPUs with multi-core CPUs in order to achieve better performance. Integration of safety-crtical applications on multi-core architectures is a hot topic both for designers and for the certification authorities [3]. The multi-core architectures provide the necessary performance increase for industrial applications, but they provide very little support for the integration of multiple safety-critical applications or mixed-criticality applications in a way that the performance gain is preserved. Specifically, a

sufficient spatial and temporal isolation of components cannot be achieved.

Time predictability. The difference between embedded industrial applications and general purpose applications is that they are often under constraint of *time*. Also, co-ordination of different tasks is necessary to achieve the time constraints. Multi-core CPUs are not designed with sufficient level of determinism due to intensive use of shared resources towards better performance. Hybrid SoC architectures can be molded into a system with high level of determinism and sufficient performance.

Power Consumption. Environmental challenges have lead the industrial community to make a significant effort in lowering the energy consumption of computer systems. Reducing the number of physical computer components in a system reduces overall power consumption. Further, selective and modular operation of a system provides more room for optimization of power consumption of individual system components. *Mixed-Criticality Integration.* As the general population is getting more and more dependent on internet based services (i.e., communication, personal organization, navigation) and as systems provide better capabilities for partitioning and segregation it is expected that non-critical functions will get integrated with safety-critical and real-time systems. The MPSoC technology offers these abilities almost by default.

Adaptation. Embedded industrial application are often deployed in harsh, inaccessible and stochastic environments (e.g., space, deep sea). It is important that the corresponding computer system is able to adapt to environment changes without loss of functionality. If an unforeseen event brings a system into a failure state, the ability to adapt can prolong the life of a system, and ensure reliability until the system is returned to a safe state. An MPSoC is set of hardware components, where a component can be used to monitor and change a state of another component without influencing the rest of the system.

Heterogeneity. A specialized hardware like a digital signal processor (DSP) or graphics processing unit (GPU) is much more efficient in handling some tasks than a standard CPU. Heterogeneity ensures efficiency by allowing applications to be mapped to the core whose execution will ensure best requirements coverage. Heterogeneity of its requirements. It also provides versatility and mixed-integration capabilities.

It is clear that all these challenges are highly connected and mutually dependent, this is why a solution must be universal, rather then ad-hoc solving one problem at the time. New technologies provide capabilities to implement alternative hardware architectures and evaluate them on real-life scenarios.

III. ACROSS MPSoC

The ACROSS MSPSoC [1] is a many-core architecture implemented completely in an FPGA. At the core of the MPSoC is a time-triggered network on chip (TTNoC). It connects eight independent host components based on soft-coded Nios2 processors. Each components (also called μ-Component) is fully functional embedded system with local memory and set of input/output peripherals (IOs). The TTNoC uses encapsulated communication channels and special linking interfaces to interact with μ-Components. Four μ-Components are used as system components performing system, configuration and maintenance functions (i.e., global time service, mass storage, monitoring services, IO services). Other four μ-Components are reserved for application software. The application components share system components and their services in full temporal and spatial isolation provided by TTNoC. The MPSoC provides two levels of of fault-tolerance: The on- and the off-chip fault-tolerance. Each component is a fault containment unit and the MPSoC is considered as error containment unit. The architecture was designed using a services-oriented approach, and it offers three types of services: core services, optional services, and application-specific services. The core services handle platform related tasks (e.g., basic communication). The optional services provide extended functionality to core services and can be modeled on a specific requirements set (e.g., security). The application specific services are user level services implemented on top of the core and optional services. The ACROSS MPSoC is implemented with a unique set of IO connections to allow cross-domain integration.

IV. Hybrid System-on-a-Chip

This paper explores the capabilities of the state-of-the-art FPGA devices and their ability to serve as an alternative to COTS multi-core processors. Specifically, a crossover device between these two technologies is being explored. In this paper it will be called a *hybrid SoCs*. The hybrid SoC is a device that combines FPGA and a hard processor on a single chip. FPGA devices are used extensively in embedded applications, especially in signal processing and communication devices. Most frequently it is used as a secondary device for specific tasks in a system. An FPGA is a programmable hardware, it can be used to emulate hardware functions or to execute software modeled in the hardware like logic. It provides a platform for prototyping of hardware or a tool for software acceleration. Recent advancements in FPGA production technology enabled significant increase in the capacity and performance of the FPGA devices. Figure 1 shows how the FPGA technology advanced in recent years, as an example we compare number of fundamental logic blocks of two major FPGA vendors and their flagship devices (Logic Elements (LE) for Altera, and Logic Cells (LC) for Xilinx). It can be assumed that other properties of FPGA devices advanced in the similar manner. This puts use of FPGAs as an alternative hardware architecture in a wider spectrum of industrial embedded applications (i.e., ACROSS MPSoC see Section III).

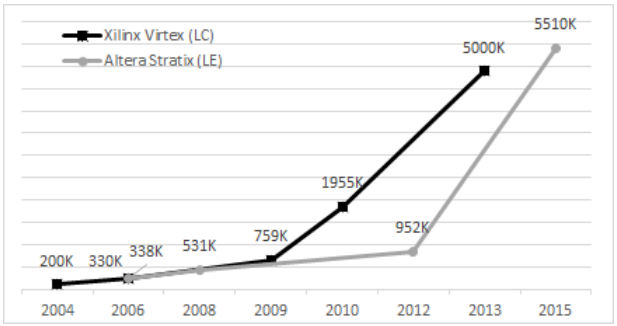


Fig. 1. Development of FPGA devices in recent years in reference to number of fundamental logic blocks [4], [5].

The rapid FPGA development trend correlates with the introduction of hybrid SoCs. The FPGAs are flexible and can be used to implement almost any function, hardware or software. However, the implementation of standard hardware architectures on FPGA is performance limited. On the other side standard hardware architectures lack flexibility. The use of both devices in an interlocked architecture is relatively common (e.g., [6], [7]). The integration of the same devices on a single chip provides more performance, lower latencies and the possibility to directly augment one or the other device and build custom architectures. Most notable architectures are Zynq from Xilinx [8], Cyclon V and Arria V form Altera [9], [2], and SmartFusion2 from MiroSemi [10].

The hybrid SoC architecture provides a new dimension for embedded designers. The advantages of combining both devices can be utilized in a joint system with a low latency. In this paper we present a custom built architecture which integrates both FPGA and CPU in a reliable and deterministic fashion. The prototype platform chosen for this purpose is Altera Arria ST SoC (see Figure 2).

A hard processor implemented on this platform is the ARM Cortex A9 dual-core processor. It is a central part of a system called a hard processor system (HPS). It also includes a direct interfaced 1 GB DD3 memory, 32KB of instruction and data L 1 cache per core, shared 512 KB L2 cache memory, QSPI flash memory and a generic set of IOs (e.g., USB, UART, Ethernet, GPIOs etc.). The second component on the chip is an Arria V FPGA. It is a medium capacity programmable fabric (460K LEs) device. The FPGA has two direct interfaced 1 GB memory modules, an FPGA specific set of control and programming devices and a generic set of IOs. The communication hub on the chip is a L3 interconnect, that connects HPS with the FPGA and board peripherals. The interconnect has three dedicated AXI bus bridges for the communication between HPS and FPGA: HPS-to-FPGA, FPGA-to-HPS and a lightweight HPS-to-FPGA. These allow mutual exchange of resources and full cooperation between two systems. The block diagram of the Altera Arria V ST SoC development board is shown in Figure 2.

How does the hybrid SoC respond to the challenges in industrial embedded applications? When it comes to performance, a recent survey reports that an embedded systems runs on $\leq$ 500 Mhz processors in average [11]. The hybrid SoC is able to match that on the HPS side and ensure even better performance. The FPGA is able to provide additional dedicated cores or components which can be used accelerate tasks. It also provides a platform to integrate diverse systems without physical or logical overlapping. This is essential for mixed-critical and cross domain systems. Clustering of systems on a single chip ensures efficient power consumption. The operation of the two systems are independent and ensures the isolation required for safety critical applications. Soft-coded components also allow full segregation even within FPGA. An other aspect of hybrid SoC which supports modern embedded applications is the capability to reconfigure one or the other system on run-time. For the future adaptive systems this enables more flexibility where even hardware configuration of the system could be completely changed during run-time. Recent reports are predicting a huge improvement in hybrid SoC technology in the coming years. This ensures longevity

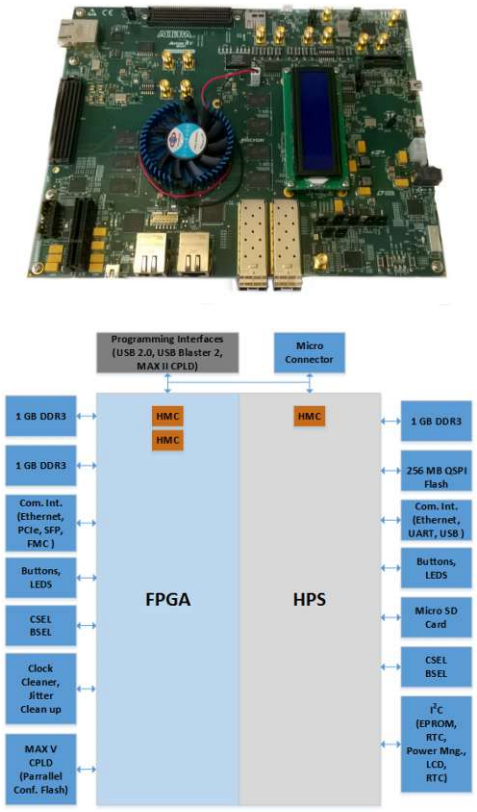


Fig. 2. Arria V ST SoC development board and a block diagram of the corresponding architecture.

of the solutions built using this technology [12].

V. Time-Triggered Multi-Processor Architecture on Hybrid SoC

The evaluation of the ACROSS MPSoC in industrial demonstrators has confirmed the predicted assumptions in achieving mixed-criticality and cross domain integration using this architecture as valid. The goal of the architecture presented in this paper is to integrate high performance components, and build a basis for a more generic setup with the focus on a modular interface defined integration of components. The design of embedded applications often includes over dimensioning to ensure longevity of a product, especially in industrial domains where the design and certification process requires extremely large efforts (e.g., avionics systems, aerospace, industrial automation). it is essential to evaluate the scaleability, portability and the platform dependence of the architecture.The integration of TTNoC-based architecture on hybrid SoC shows promising results for a large spectrum of industrial embedded applications.

The ACROSS legacy architecture was designed to host a specific set of applications and provide a specific set of services. A number of these services are platform dependent, so porting the whole architecture was unpractical. To translate the architecture to the new platform a number of the components needed to be adapted. The communication backbone remained unchanged just with slight modifications to conform

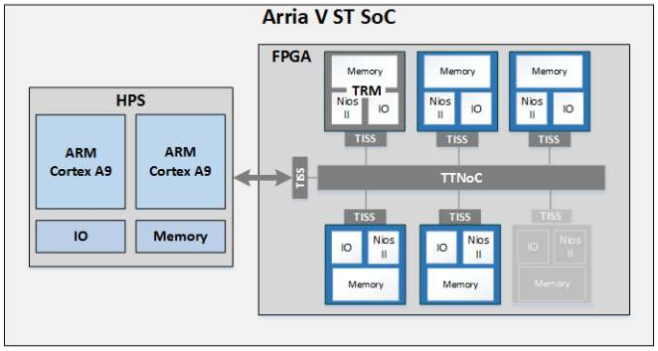


Fig. 3. Block diagram of the deterministic MPsoC architecture on hybrid SoC.

the platform and current programming toolchain. The essential parts of the communication backbone are: TTNoC, trusted interface subsystems (TISSs) and trusted resource manager (TRM).

The TTNoC consists of *fragment switches* that enable routing of traffic and global time propagation. Each fragment switch is a module with four bi-directional channels for messages and additional signals for propagation of a system wide global time. The fragment switches are combined together to create a network-on-chip. The size of the NoC depends on the number of fragment switches and it is bounded.

The TISS is a communication interface between the TTNoC and a μ-Component. Although, it is basically a part of a μ-Component, it is required if the components is to be integrated in the TTNoC. The TISS is connected with the CPU of a μ-Component via dual-ported memory, local to the μ-Component, and interrupt signals for task triggering. It also provides an instance of the system-wide global-time for the μ-Component. The global time is an essential property of a deterministic architecture, as it enables time-triggered communication, coordinated execution of tasks and improved fault-tolerance capabilities for the components [13]. The TISS is also the host for the configuration files necessary for time-triggered communication and task triggering. These can be only accessed and modified by the TRM component.

The TRM is the μ-Component responsible for maintenance and configuration of TTNoC and corresponding TISSs. It contains a tick generator which provides the basis for the system wide global time. The second important task of the TRM is the configuration of TISSs. The TRM is the only component allowed to change the state of the configuration memory of a TISS. All the soft-coded components were completely rebuilt and integrated into the MPSoC. The goal of the current implementation of the MPSoC is to create a basis for a platform and application independent architecture. Namely the intention is to fully utilize Altera's system integration tool QSyS [14] and build a system in a fully modular way. The Nios2 components are therefore designed on a generic model with clearly defined interfaces. This allows easy adaption of the components to service or application specific requirements, and simple integration of new components to the system.

One of the advantages of the TTNoC architecture is the ability to connect heterogeneous components. The CPU of the component must be able to interface the dual-ported memory and receive interrupt signals. The Arria V ST SoC allows direct coupling of IP modules implemented on FPGA with the HPS system over the aforementioned interfaces (see Section IV). This enables direct integration of the HPS with the rest of the MPSoC. The new component is also represented as a single module in the integration tool, although it is composed from building blocks located on both on FPGA and HPS. The coupling requires two interfaces one for the data transfer, or interfacing the local port memory and the other one for the routing of interrupt signals. Figure 4 shows the HPS component and the way it is connected to the TTNoC over TISS.

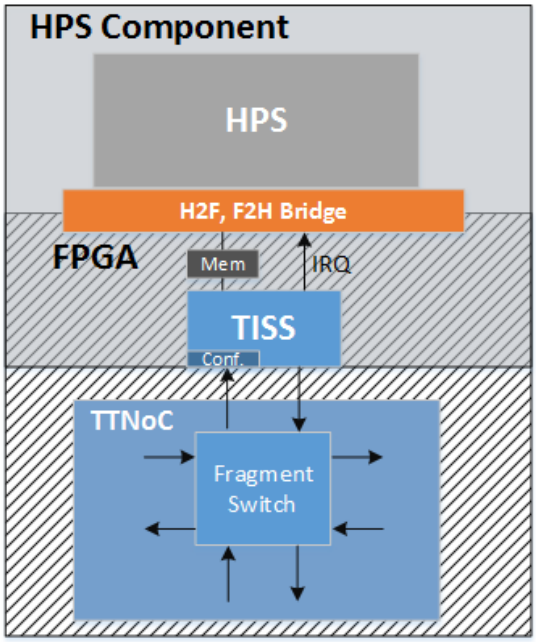


Fig. 4. Block diagram of the HPS connection to the TTNoC

The heterogeneous approach increases fault tolerance capabilities by adding another layer of fault/error containment within a chip. Although they are physically parts of the same chip HPS and FPGA modules are fully independent. On the FPGA there is a soft fault containment between μ-Components. The integration of the HPS component adds another fault tolerance mechanism to the whole architecture. The isolation between HPS and FPGA enables error containment units on both devices. The mechanisms built in the platform allow reconfiguration of individual devices if the other one fails. The MPSoC is capable of phase synchronized operation with another MPSoC. This can be used to replicate components and provide off-chip fault tolerance.

The architecture currently uses a static configuration as it is considered as a basis for certification. Next step in this direction would be to explore possibility of dynamic adaptation and reconfiguration. The hybrid SoC allows reconfiguration of the SoC during run-time, this ability can be utilized by the overlaying MPSoC architecture. Further, the HPS component provides new possibilities but it also adds to the complexity of the system. The HPS has relatively complicated memory

hierarchy and it is essential to map an application based on its safety requirements and fault hypothesis with the capabilities of the HPS component. This is also important for standard multi-core architectures. Other applications of HPS component include a runtime monitoring and verification, which are essential for the dynamic adaptation and reconfiguration.

The initial tests of the proposed hardware architecture showed successful integration of the above mentioned components with the TTNoC. This includes functionality test of individual hardware components. The integration of HPS component in the MPSoC is successful. The modular approach provides ability to extend or reduce the MPSoC's number of components in an simple and efficient way. The first step was to rebuild and adapt the hardware components on to new platform and implement HPS component, also to ensure the functionality of the communication backbone. Additional results will be provided in an extension of this paper, as a support software, applications and evaluation are still in progress.Next chapter provides a short overview of the architecture's implications on two industrial use cases.

VI. Industrial Embedded Systems Case Study

The motivation behind the work presented in this paper comes from industrial embedded applications. The spectrum of applications for the proposed architecture is wide as it is shown in [1]. In this section provide an overview of a couple of use cases, and discuss how they can benefit from the properties of the presented architecture.

1) Automotive control units (xCU): The automotive industry is represented with a high number of units per year and it needs to be in a continuous state of innovation in order to ensure an economic progress. A recent reports show the automotive industry produced about 90M vehicles last year[15]. A survey shows that major drivers behind the innovation in the automotive industry are fuel efficiency and safety. The same survey reports that four important upcoming innovations in the industry are: electric and hybrid power-trains, internet communication, car-2-car communication, car-2-oem communication, and predictive consumer analytics [16]. How does this reflect on the embedded systems within automotive industry? The conclusion is that the these systems should be able to host multiple applications from different safety zones, and ensure a required level of safety for all of them. How does the proposed architecture answer to this challenge? It is capable of running both safety critical and non-critical applications at the same time. The HPS component provides more than adequate performance capabilities for the present day applications and and future applications. It provides a specific fault hypothesis model two layers of on-chip fault tolerance, and a promising approach for off-chip architecture. The work presented in [17] shows how the similar architecture can be used as an automotive xCU. The clustering of the xCU on an MPSoC is shown to be an efficient alternative to implementation of automotive applications on high-performance COTS multi-core architectures. Clustering of xCU in complete isolation on MPSoC provides obvious advantages. For example, consuming less physical space reduces weight and power consumption. Which in effect reduces fuel consumption and increases overall effectiveness.

2) Space on-board computers: Contrary to the automotive example the space sector has low unit numbers and completely different innovation focus. The production of space equipment requires high amount of efforts, due to harsh environment and limited ability to perform repairs and changes. The applications in space are more mission critical then safety critical. Nevertheless, they face similar problems when it comes to hardware architectures [18]. Both, seek to reduce number of on-board computers, increase performance capabilities at the same time, and ensure energy efficiency. The space applications also require high adaptation capabilities. The costs of deploying and producing such a system are large, therefore the lifespan of the product must be substantial, and all components must be fault-tolerant. The mechanisms like adaptation provide the ability to prolong the life of the system by enhancing it or reducing its functionality (e.g., by removing non-essential parts a system can perform basic functions for a longer period). The proposed architecture offers integration of multiple applications on a single chip, with several layers of fault-tolerance. The hybrid SoC ensures ability to change the configuration if necessary of each device on runtime by the other one. The heterogeneous integration is also very important for the space sector, as its applications operate with large amounts of physical signals that can benefit from dedicated hardware components.

VII. Future Work

At this stage the work presented contains a hardware architecture and rudimentary software support. In this chapter we present some of the additional actions included in our future work on this topic. As stated in the introduction the goal is to evaluate scaleability of the system by experimenting with different setups of the architecture. The heterogeneous approach is already integrated in the architecture, but there is still room to expand it by integrating additional components based on other CPUs. One of the concrete examples is the integration of the Leon3 SPARC soft-coded processor from Geisler [19]. The Leon3 has a fault-tolerant version and it is highly utilized in space domain embedded systems. The second goal is to ensure efficient portability to future generations of the platform as well as the platform portability. This would contribute to a more general use of the architecture.

VIII. Related Work

The MPSoC related research has produced a high number of publications in recent years. With the introduction of hybrid MPSoC the availability of these systems increased dramatically, and so has the interest of the community. The research on MPSoC is also the vocal point of several major research initiatives in EU. Projects ACROSS [20][1], MultiPartes [21], ARAMIS [22], EMC^2 [23] are all investigating the introduction of multi-core and MPSoC architectures in the domain of safety critical and mixed-critical systems.

The automotive industry is a highly interesting domain for MPSoC platforms.The author in [24] provides a demonstration of vehicle powertrain implementation using Xilinx Zynq [8] platform. Although it is a dedicated solution for a single application it is still showing the extent of the MSPoC's capabilities. The authors in [25] provide a mapping of AUTOSAR [26] to the MPSoC based architecture. Another example of MPSoC application for automotive use case is presented in [27]. The

paper provides a framework for dynamic allocation of task to MPSoC components. It can be noticed that the hybrid SoC is most frequently used as application oriented systems. The capacity of the hybrid MPSoC to be used as generic platform for industrial embedded systems is still to be explored.

The use of hybrid MPSoC applications stretches on multiple industrial domains. The authors in [28] give a survey on usability of the hybrid SoCs in the avionics domain, specifically on the Xilinx Zynq platform. In [29] the authors explore the capabilities of the hybrid MPSoC platform to perform dynamic partial reconfiguration. These are only some of the works focused around hybrid MPSoCs. The direction of development of major hardware vendors confirms this statement [12] [30].

IX. Conclusion

This paper presented a heterogeneous MPSoC architecture featuring a time-triggered network-on-chip implemented on a hybrid SoC platform. The advantages of the hybrid SoC are used to enhance an an earlier version of the MPSoC with the hard processor component built on ARM Cortex C9 processors. In addition to the HPS Component other novel aspects are included in the process. A generic modular design as the basis for model oriented development, adaptation and dynamic reconfiguration. We discussed the benefits of this architecture, on a couple of use cases, and shown the potential of this approach. The hybrid SoC technology is becoming more available every day, and its capabilities are increasing accordingly. The proposed MPSoC uses this technology to build an architecture capable of hosting industrial embedded applications with even most demanding requirements.

Acknowledgment

This research has received funding from the ARTEMIS Joint Undertaking (JU) under grant agreement n° 621429. The authors would like to acknowledge the work performed in the project founded by ARTEMIS JU under the ID ARTEMIS-2009-1-100208 (project ACROSS). Also this work is made possible due to generous contribution of TTTech, as they granted access to their IP essential for the work presented in this paper. The responsibility for the content rests with the authors.

References

[1] C. Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek, "The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, Sept 2012, pp. 105–113.

[2] Altera. (2015) Arria V FPGAs and SoCs. [Online]. Available: https://www.altera.com/products/fpga/arria-series/arria-v/features.html

[3] Certification Authorities Software Team (CAST). (2014) Position Paper CAST 32. [Online]. Available: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf

[4] Altera Inc. (2015) Website. [Online]. Available: https://www.altera.com

[5] Xilinx Inc. (2015) Website. [Online]. Available: http://www.xilinx.com

[6] MPC8641 PowerPC I/O Processorwith Virtex-4 FPGA - VME/VXS. (2011) Pantek. [Online]. Available: http://www.pentek.com/deliver/deliver.cfm?DI=1&FN=4207bro.pdf

[7] 2nd Generation Intel Core i7 Processor with Kintex FPGA - Model TIC-INT-VPX3a. (2013) Pantek. [Online]. Available: http://www.elma.com/products/systems-solutions/embedded-boards/product-pages/single-board-computers/2nd-generation-intel-core-i7-processor-with-kintex-fpga-tic-int-vpx3a-detail/

[8] Christian Kohn, Xilinx. (2015) Cyclone V SoCs. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1231-partial-reconfig-hw-accelerator-vivado.pdf

[9] Altera. (2015) Cyclone V SoCs. [Online]. Available: https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html

[10] MicroSemi. (2015) SmartFusion2 SoC. [Online]. Available: http://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion2

[11] UBM tech. (2013) Embedded Market Survey. [Online]. Available: http://e.ubmelectronics.com/2013EmbeddedStudy/index.html

[12] Altera Inc. (2015) Roadmap. [Online]. Available: https://www.altera.com/products/soc/roadmap.html

[13] H. Kopetz, "Genesys-a cross-domain architecture for dependable embedded systems," in *Dependable Computing (LADC), 2011 5th Latin-American Symposium on*. IEEE, 2011, pp. 1–6.

[14] Altera Inc. (2015) QSyS. [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/quartus-ii-subscription-edition/qts-qsys.html

[15] OICA. (2015) Production Statistics for Motor Vehicels. [Online]. Available: http://www.oica.net/category/production-statistics/2014-statistics/

[16] KPMG. (2015) KPMGs Global Automotive Executive Survey. [Online]. Available: https://www.kpmg.com/Global/en/IssuesAndInsights/ArticlesPublications/global-automotive-executive-survey/Documents/2015-report-v1.pdf

[17] A. Wasicek, O. Höftberger, M. Elshuber, H. Isakovic, and A. Fleck, "Virtual can lines in an integrated mpsoc architecture," in *Proceedings of the 17th IEEE Computer Society Symposium on Object/Component/Service-oriented Real-time Distributed Technology (ISORC)*, June 2014. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/1060.html

[18] L. Berrojo, R. Moreno, R. Regada, E. Garcia, R. Trautner, G. Rauwerda, K. Sunesen, Y. He, S. Redant, G. Thys *et al.*, "Scalable sensor data processor: A multi-core payload data processor asic," in *ESA Special Publication*, vol. 732, 2015, p. 6.

[19] Geisler. (2015) Leon3 Sparc. [Online]. Available: http://www.gaisler.com/index.php/products/processors

[20] ARTEMIS JU. (2015) ACROSS. [Online]. Available: http://www.across-project.eu/

[21] FP7. (2015) MultiPARTES. [Online]. Available: http://www.multipartes.eu/

[22] (2015) ARAMIS. [Online]. Available: http://www.projekt-aramis.de

[23] ARTEMIS JU. (2015) EMC2. [Online]. Available: http://www.artemis-emc2.eu/

[24] M. Dendaluce, "System-on-Chip-based highly integrated Powertrain Control Unit for next-generation Electric Vehicles: harnessing the potential of Hybrid Embedded Platforms for Advanced Model-Based Control Algorithms," in *The 28th International Electric Vehicle Symposium and Exhibition*, 2015.

[25] M. Urbina and R. Obermaisser, "Multi-core architecture for autosar based on virtual electronic control units," in *Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on*, Sept 2015, pp. 1–5.

[26] S. Fürst, "Autosar–a worldwide standard is on the road."

[27] J. Rettkowski, P. Wehner, M. Schülper, and D. Göhringer, "A flexible software framework for dynamic task allocation on mpsocs evaluated in an automotive context," in *Applied Reconfigurable Computing*. Springer, 2015, pp. 255–266.

[28] S. VanderLeest and D. White, "Mpsoc hypervisor: The safe amp; secure future of avionics," in *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, Sept 2015, pp. 6B5–1–6B5–14.

[29] A. Beasley, L. Walker, and C. Clarke, "Developing and implementing dynamic partial reconfiguration for pre-emptible context switching and continuous end-to-end dataflow applications," in *Altera SoC Developers Forum*. University of Bath, 2015.

[30] Xilinx Inc. (2015) Website. [Online]. Available: http://www.xilinx.com/about/generation-ahead-16nm.html

CHAPTER 5

A Mixed-Criticality Integration in Cyber-Physical Systems:A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform

Solutions for Cyber-Physical Systems Ubiquity

Norbert Druml
Independent Researcher, Austria

Andreas Genser
Independent Researcher, Austria

Armin Krieg
Independent Researcher, Austria

Manuel Menghin
Independent Researcher, Austria

Andrea Hoeller
Independent Researcher, Austria

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Published in the United States of America by
IGI Global
Engineering Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Druml, Norbert, 1980- editor.
Title: Solutions for cyber-physical systems ubiquity / Norbert Druml, Andreas Genser, Armin Krieg, Manuel Menghin, and Andrea Hoeller, editors.
Description: Hershey, PA : Engineering Science Reference, [2018] | Includes bibliographical references.
Identifiers: LCCN 2017012032| ISBN 9781522528456 (hardcover) | ISBN 9781522528463 (ebook)
Subjects: LCSH: Cooperating objects (Computer systems)--Handbooks, manuals, etc. | Internet of things--Handbooks, manuals, etc. | Automatic control--Handbooks, manuals, etc.
Classification: LCC TK5105.8857 .H367 2018 | DDC 004.67/8--dc23 LC record available at https://lccn.loc.gov/2017012032

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

For electronic access to this publication, please contact: eresources@igi-global.com.

Chapter 7
A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform

Haris Isakovic
Vienna University of Technology, Austria

Radu Grosu
Vienna University of Technology, Austria

ABSTRACT

A Cyber-Physical System (CPS) describes a system or a system-of-systems closely and actively coupled with environment. It comprises the digital intelligence system, a co-dependent physical system (i.e., electrical, mechanical) and the system environment. Since the beginning of modern computer systems integration was ever present challenge, from the huge single room computers to the IoT. Today applications interleave and build larger systems with different system requirements and properties. Implementation of safety critical applications together with non-critical applications within the same platform is almost inevitable in modern industrial systems. This article provides a retrospective overview of the major integration challenges and the current problems in mixed-criticality environments. Finally, it provides an insight in a hardware solution which creates deterministic platform for mixed-criticality applications.

1. INTRODUCTION

The topic of cyber-physical systems (CPS) is an interdisciplinary subject that connects computer engineering and computer science with other disciplines like electrical, mechanical, chemical and bio-engineering. In essence, it provides a shell for the system design which binds digital and physical world in a compact methodological form. It provides a better understanding of the system from various perspectives, it increases time to market and overall efficiency of the product. This concept was crafted

DOI: 10.4018/978-1-5225-2845-6.ch007

as a response to the increasingly complex structure of modern systems. The advantages of computer driven physical systems were recognized in the early stages of the computer development timeline. The physical systems controlled by a computer would reach their physical limits much rapidly, they were getting upgraded more frequently and in conclusion getting more complex. The initial approach of the system design was highly segregated in means of engineering disciplines. For the most part physical components of a system would be designed according to the techniques and methods used in a specific discipline, and then interfaced with a specific computer system. This approach provided a relatively clean design methodology, but as the systems got more complex it was more difficult to ensure that the functional properties of the system conform to the system specifications and regulatory guidelines. The work presented in this chapter reflects on the challenges present in the domain of mixed-criticality systems and the overcoming need for seamless integration. It also offers a viable solution based on a time-triggered architecture implemented on a hybrid system-on-a-chip platform. This novel approach to create a modular configurable deterministic hardware architecture combats core problems of industrial computer systems, where safety critical applications interact and operate under same conditions as non-critical applications. This closely coupled relation is extremely complicated on a commercial off the shelf (COTS) hardware. It is also highly expensive in terms power consumption, performance and utilization. The presented architecture enables clear separation between different tasks in space and time without performance loss. The extended plans foresee a tool chain integration to increase the ability to build an application directly from hardware level up.

1.1 Chapter Outline

This chapter provides a short reflection on mixed-criticality integration in cyber-physical systems. It explores the challenges and basic properties for the seamless integration not only user applications, but also underlying platform components, hardware, software and the physical environment. Section 1 gives a short introduction in cyber-physical systems and mixed-criticality integration, in particular Section 1.1 gives a short historical summary of the cyber-physical systems and turning points that lead to the modern state of CPS. Also, Section 1.3 gives an overview of the major challenges or objectives for mixed-criticality integration. Further, Section 2 gives a brief introduction in the background knowledge on the relevant topics. First, Section 2.1 describes spatial and temporal isolation as vital properties in the design of systems for mixed-critical integration. Section 2.2 introduces an innovative computer architecture that combines a hard-coded computer processing units with an FPGA device on a single chip, it is a synergy of different approaches and a product of merging knowledge from two different directions. Further, Section 2.3 describes an architecture built for mixed-criticality integration based on time-triggered communication and FPGA technology. Section 3 provides a description of the architecture that merges technologies described in Sections 2.2 and 2.3. Section 4 offers a brief overlook on the presented architecture and its ability to meet the challenges described in Section 1.3. The ability of the architecture to resolve practical issues has been shortly visited in Section 4.1. Finally, Sections 5 and 6 provide future work potentials and closing thoughts.

1.2 A Historical Overview

- **The Origins:** The name cyber-physical system was derived from the term "cybernetics", which is used to describe systems that used closed-loop feedback control. The term was introduced by an extraordinary 18th century physicist and mathematician A.M. Ampere (Ampère, 1838), and later revived by a 20th century mathematician and pioneer of modern control theory Norbert Wiener (Wiener, 1961). However, use of the term cyber-physical system was adopted as recently as 2006, it is used as an umbrella for all computer controlled systems that in one or the other way interacted with the physical world around it.
- **The Computer:** The first computers were originally created with a purpose of performing complex calculations, they were big machines installed in large rooms with highly complex structure and handling. As the computers became more accessible and affordable their services were adopted in other disciplines i.e., manufacturing, space, robotics, and consumer appliances.

In the 1950s computers were first applied in a control of industrial equipment, in the process called computer assisted manufacturing (CAM) using Automatically Programmed Tools (APT) language, it was a system designed to control a milling machine with a help of a computer (Ross, 1978). These machines were forerunners for the modern computer numerical control (CNC) machines and computer aided manufacturing. However, 1960s is the time when the age of computers really began, more and more fields in civil and military domains started using computers for various purposes. In 1961 a first mass-produced industrial robot UNIMATE was integrated in the production lines at General Motors (Nof, 1999). Around the same time, NASA included first digital computers in space programs Gemini and Apollo (NASA, 2016). This was followed by an introduction of the first commercial minicomputer DEC's PDP-8 (DEC, 1967). In the 1970s the development of computerized systems was highly accelerated and commercialized, new actors in computer technology development (i.e., Intel, Microsoft, HP, etc.) emerged and set the course for the next few decades.

- **The Microprocessor:** With the development of micro processing a huge spotlight was set computer architectures for personal computing, bringing computer aided development in completely new areas and disciplines. The micro processing found its way in other applications like control of fuel injection in cars or handheld calculators. The whole surge of new application was followed by a massive change in software as well, new computer languages like Pascal or C were introduced. (Wirth, 1971) (Kernighan & Ritchie, 1978). There was also a strong breakthrough of mass networking systems with Teletext information system and research-oriented network ARPAnet (BBC, 2012) (O'Regan, 2012).
- **The Embedded World:** The development of micro-controllers in 1980s enabled mass utilization of embedded computer systems in new applications, from industrial machines and communication devices to consumer electronics, and entertainment systems. In 1981 a first prototype for the direct drive arm was proposed, it shared the same basic principles used in industrial robots today (Asada & Kanade, 1983). An introduction of small-sized computers and networking breakthroughs influenced put high accent on communication disciplines. In 1983 Bell Labs introduced a first mobile

phone communication standard called Analogue Mobile Phone System (AMPS). It was embodied in a first commercial mobile phone Motorola DynaTAC 8000X (O'Regan, 2012). At the same time, multi-annual research efforts to build more flexible hardware resulted in the development of first Field-Programmable Gate Arrays (FPGAs). In 1985 Xilinx released their first FPGA device (Trimberger, 2012). It is a programmable logic device that can be customized for a specific application by a user. It combines logic blocks with programmable interconnects and multiplexed sets of IOs. The FPGAs changed the view on product development, allowed high integration capabilities and faster time-to-market (Carter, 1994). The development of FPGAs remained one of the most vibrant fields in computer science and electronics.

- **The Network:** The development of mobile phones was followed by networking revolution and introduction of OSI model, and consequently the invention of World Wide Web by Tim Berners-Lee. The interaction between systems and distributed computing became critical topics in industrial and scientific circles. The distributed computing was a significant change to the legacy systems that operated completely in isolation. At this point computers were already on multiple levels: large mainframe and server computers, personal office or home computers, and embedded or small-scale integrated computers. The high level of interaction between individual units had been already available for the first two groups, however, it was still relatively new in the scope of embedded systems. The distributed embedded systems yielded the term "Feldbus", it symbolizes a network of local field level devices with a focus on reliability, availability and safety of the system. They provided the ability to control processes efficiently and in real-time (Kopetz, et al. 1989). Throughout 1990s trends of miniaturizing chips and increasing connectivity among system continued. The personal computers became a common tool in both business and private sector (see Figure 1). The number of networked and distributed embedded systems increased immensely, for example a number of mobile phone subscriptions had an average growth of 36% of new subscribers per year over the period between 1984 and 2003 (see Figure 2).
- **The Model:** The initial computer controlled systems were built to explore the capabilities of this new technology and with focus on basic functions. In most cases, they represented ad-hoc solutions with a low dependability level. A major challenge at the time was to ensure reliability of the systems even for the applications that required strict safety insurance levels. The challenge of simply creating efficient computer system and applying it was not the largest problem anymore. How to make it safe and reliable for all applications was a question more difficult to answer (Bowen & Stavridou, 1993). At the same time, there was a strong shift towards knowledge based or model based design of systems in computer engineering. A model based design reduces building efforts, system testing and maintenance efforts of complex systems (Studer, Benjamins, & Fensel, 1998). A first version of Common Object Request Broker Architecture and Specifications (CORBA) was released in 1991, and up to these dates it represents a state-of-the-art framework for integration of heterogeneous systems (Vinoski, 1993). The networking increased efficiency of the system in a number of ways, but it created a huge space for other obstacles in system design and operation i.e., synchronization and coordination, security and integration. A guideline of security standards and principles by NIST in 1995 identifies integration as one of the basic elements in the design of secure systems (NIST, 1995). The integration understands connecting systems and subsystems using hardware, software or physical interfaces such that they operate as a single functional unit, providing emerging functionalities that are impossible to implement using individual components.

Figure 1. Number of mobile cellular subscribers in United States per 100 people: 1984-2003
The World Bank Group, 2016.

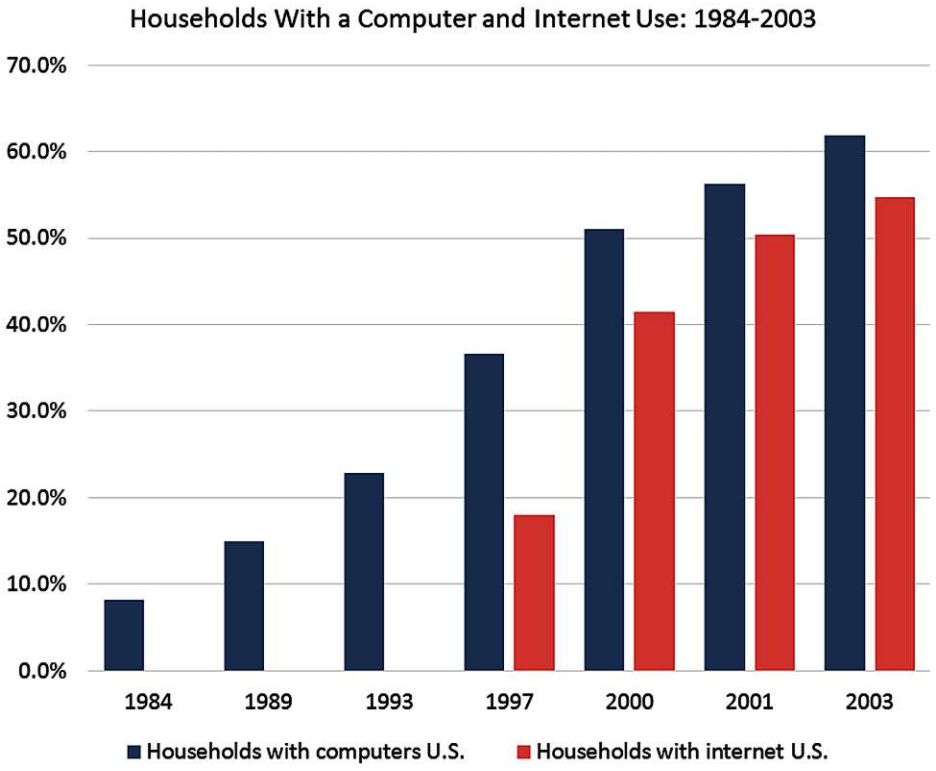
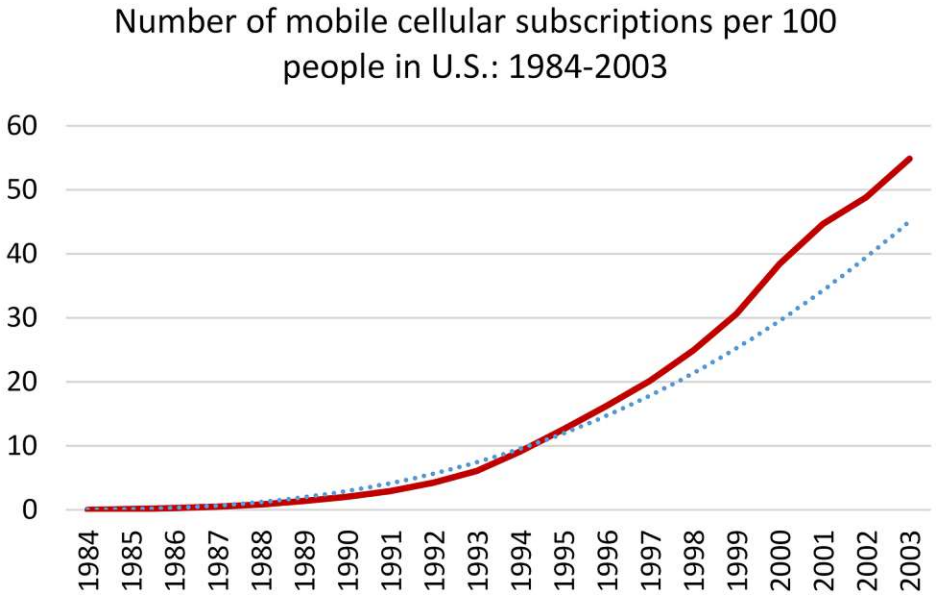


Figure 2. Number of mobile cellular subscribers in United States per 100 people: 1984-2003
The World Bank Group, 2016.



- **The Integration:** The technologies that followed were designed with seamless integration as one of the basic architectural requirements. Despite the fact that mobile telephony, used means of wireless communication since 1980s, the traditional cable operated computer networks officially migrated to wireless communication in 1997 with the release IEEE 802.11 standard (IEEE, 2016). It also stimulated industrial driven research and development of various wireless sensor networks. The service-oriented architectures, multi-core and system-on-a-chip (SoC) architectures represent a next step in the evolution of computer systems. The first originated in mainly in the domain of the Internet, and then transcended to industrial applications over time (Blomstedt, et al. 2014). The idea of multi-core architectures was present for a long time, however it was limited by the chip production technology. Increasing performance of single-core chips also increased their complexity, to manage this instead of increasing performance of the single-core the developers find a way to multiply cores on the chip. Moreover, the SoC principle was already implemented in microcontrollers and the improvement of chip production technology allowed this technology to scale up and implement high performance systems on a single chip. The introduction of multi-core chips was mainly driven by performance, where the SoCs were designed with priorities on efficiency and economic integration with focus on embedded devices and industrial applications.

1.3 Challenges for Mixed-Criticality Integration

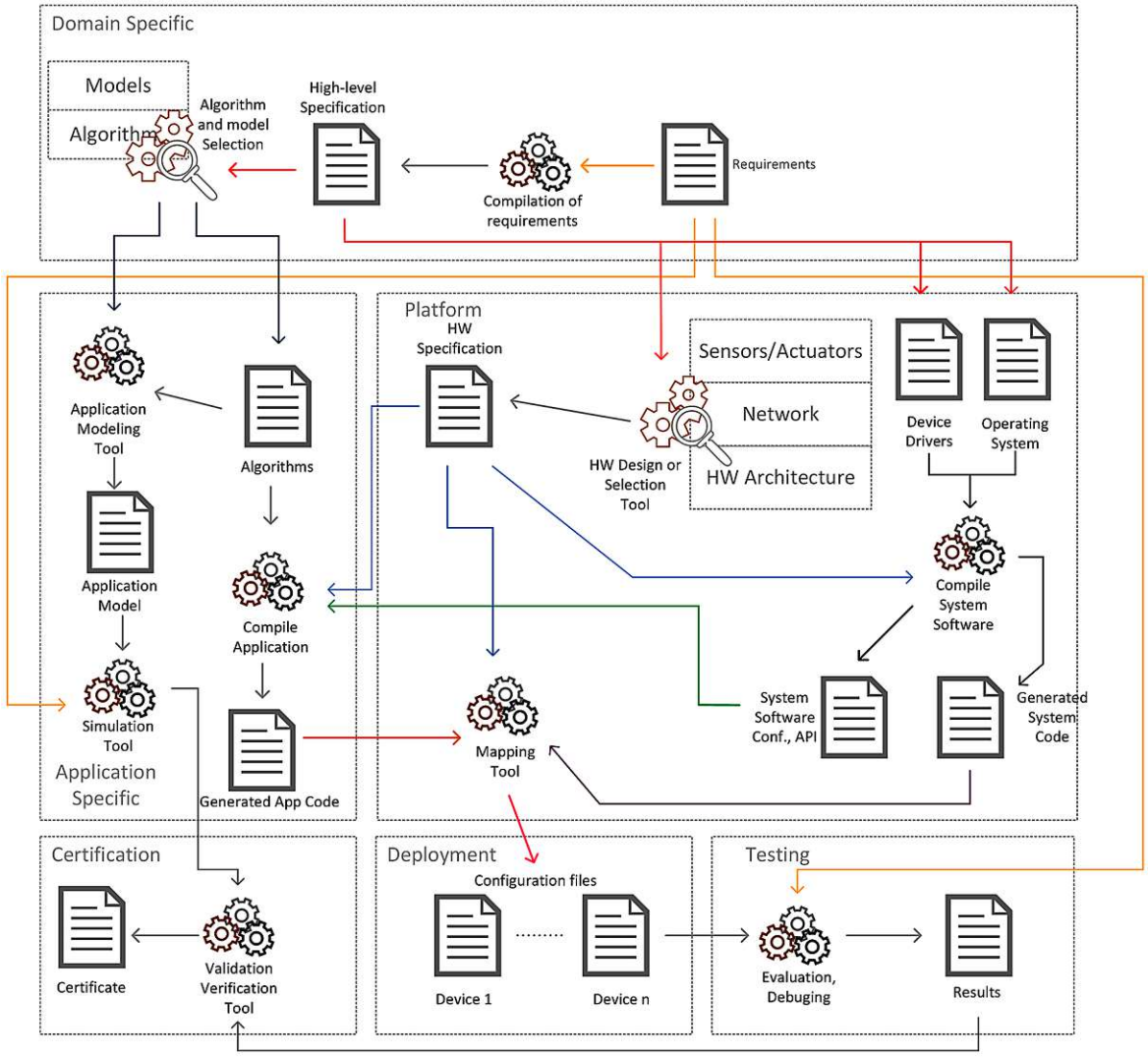
Although mixed-criticality question was forced from a safety perspective, it can be easily projected on other system properties i.e. security, fault-tolerance, performance etc. At the same time, it represents the ability of the systems to perform multiple tasks with different priorities and a way to perform system optimization. It is basically an integration/optimization problem, merging multiple components (e.g., electronic control units (ECUs) of a car) of a system in a single functional unit with emerging benefits in the likes of cost, performance or power consumption.

- **Performance:** When it comes to performance the computer systems surpass current demands of industrial applications. As an example, modern multi-core processors are capable of serving multiple applications without using maximal capacity. However, they are lacking in efficiency and reliability and this is why industrial applications, especially safety critical applications, still heavily rely on single-core processors and SoCs. Moreover, the single-core processors are being outgrown by the state-of-the-art applications. The chip manufacturers are abandoning single-core production and turning towards more profitable products. The numbers of non-essential applications (e.g., smart phone car control) and smart technologies (e.g., advance driver assistance system (ADAS)) are increasing rapidly. Having a safest or most reliable cars are not necessarily deciding factors anymore when selling or buying a car. A modern user experience and comfort inspired by other fast growing consumer electronics products translate also in automotive and other industry domains, traditionally more concerned with properties i.e., safety, security and reliability. Integration of all these emerging functions together with basic essential functions is a challenge.
- **Determinism**: The functions are divided into tasks, each task has a different set of requirements (e.g., real-time, safety, security). Different tasks have different priorities and the system must ensure that the functions with highest priorities (e.g. braking system, engine control) perform their tasks uninterrupted. At the same time the system must provide best efforts to ensure that all functions are available. Multi-core architectures use complex and nondeterministic memory

hierarchies and inter-core communication, and highly improbable to provide sufficient guarantees that a high priority task will finish its execution on time and not get interrupted by any other action. Single-core platforms are slowly being retired, as they are becoming obsolete in terms of performance and efficiency. However, they are still the best option for a deterministic hardware architecture. A big question is how to ensure needed performance and ensure the necessary level of deterministic behavior for different applications or sets of applications?

- **Optimization:** A way to deal with the above-mentioned issues, is to increase the number of computer units in a system and provide complete hardware and software isolation of different functions. This would enable an increase of system functions and maintained the classic approach on cross-critical integration. On the other side, it would increase the complexity of the system and reduced overall efficiency. A good example is the automotive industry. The number of electronic control units (ECUs) in an average personal vehicle almost doubled in the last ten years and it currently ranges between 50 and 100 ECUs, depending on the size and class of the car (Johnson, Gannamaraju, & Fischmeister, 2015). Alone weight and power consumption of those ECUs and the wires that connect them are significant factors in the fuel economy of a car. The efforts required to ensure error-free operation from design to testing is also a significant factor in the overall economics of a vehicle. It is clear that merging individual ECUs in "larger" mixed-functionality units could result in reduced overall weight, power consumption, complexity, and space. An additional aspect is the development process these systems. Figure 3 provides an illustration of a development process in CPSs, although it provides just a portion of an actual development procedure it demonstrates the complexity level of the task. It usually includes several often-incompatible tools for collecting and verification of requirements, for design and configuration of hardware, system software, communication, applications, and for analysis, testing and certification of the system. The optimization of systems is equally challenging and important as optimization of the design and development process of these systems.
- **COTS:** Designing a hardware platform specifically for an individual application or a set of functions is must be economically justified. The number of manufactured and sold units must justify the amount of efforts to build such a system. Such platform provides the exact intended functionality, but it enables very little flexibility and it requires significant costs compared to COTS solutions. On the other hand, COTS solutions are generic and they are designed for multiple purposes, which means that they are rather limited when it comes to specific niche applications. One of the major challenges is how to build a system platform flexible enough to integrate multiple domain applications and combine functions with different levels of criticality using generic COTS platforms?
- **Taxonomy:** The classification of systems according to stakeholder requirements and system properties is another large issue. The spectrum of system requirements depending on the domain and application type can be vast. System properties i.e. Reliability, safety, security is complementary, however, one doesn't necessarily guarantee the other. The reliable system doesn't always represent a safe system, and unreliable system can be safe (Rushby, 1994). Similar conjunctions can be established with other properties, so it is important that a system is correctly classified from the start. Different properties have different levels of criticality, they also have different domains of implementation. Some properties are implemented in hardware, where others are provided by a software program or realized using mechanical or electrical mechanisms. If the system is defined

Figure 3. An abstract illustration of a design and development process in CPSs



appropriately in all domains, unnecessary redundancies could be avoided, thus saving efforts and reducing cost.

- **Modeling**: A CPS is a set of systems co-dependently operating in different domains. On one side, there is an analog domain, described using continuous mathematics, it represents the physical world in CPS. On the other side, there is a digital domain. It uses discrete mathematics and it represents the computer realm. Models are used on both sides to describe the system and to provide designers with a chance to evaluate its basic properties before they actually start building it. A number of modeling tools and languages used on both sides is enormous. To ensure interoperability, traceability and verification between different models is a task that needs to be undertaken at multiple levels of in the CPS design process. It is one of the crucial requirements for mixed-critical integration in CPS.

2. BACKGROUND

A single hardware platform to host multiple functions using software based separation solutions could be an optimal solution for cross application and cross domain integration. This needs to be supported throughout the design process of a system, from the models of the physical part to computer hardware and software. The SoC platforms have emerged as a bridging solution for this problem. They are able to integrate multiple processors with different performance characteristics, together with other components (e.g., memories, peripherals) which can be tailored for a specific domain, specific application or a specific set of functions. Moreover, a new generation of SoC that integrates a hardcoded processor with flexible FPGA devices are being utilized to create platforms for development of CPSs. They ensure a sufficient level of dynamicity where at the same time are able to provide enough performance to host state-of-the-art industrial and consumer oriented applications.

2.1 Spatial and Temporal Isolation

Mixed-criticality integration relies on a group of basic system properties. Most of all, separation or isolation of individual tasks or functions must be guaranteed. Each task must be ensured that during its operation all required resources are available. A given task can be a simple software computing routine which requires only computing resources i.e., processor and corresponding memory infrastructure. Depending on constraints of the given task the extent of the resource's availability and readiness level it's determined. If the task is hard real-time, the result of the tasks must be delivered in an only in a given timeframe. If the task were isolated, implemented alone on a single processor, with almost private memory, it would be easy to reserve a time and a place for it to perform its duties. However, real systems and the corresponding tasks are generally more complex. They control functions that rely on a complex chain of codependent tasks, a series of peripheral devices for input/output purposes and communication, and sensors and actuators for interacting with the environment. To ensure causality of events and accuracy of values of different variables all components need to be separated and integrated at the same time. System resources need to be shared between the individual tasks and functions, and these need to be isolated in the system both in space and time. A spatial isolation guarantees that a task or a function can occupy a system resource or set of resources (e.g., memory, processor, communication channel) without interferences of any other activities in the system. A temporal isolation guarantees that the execution of any task cannot influence the temporal behavior of any other task or function, and that the execution time of the task cannot be influenced by the execution of any other task or function.

Most frequent problem for the spatial isolation is memory management. In the modern systems, the size of the memory and number of memory accesses is extremely large what makes the duty of managing them highly complex. It takes a series of hardware components and specialized software routines to manage memory during the execution. The memory is a storage device organized in cells and each cell is marked with an address, this address also called a physical address. The memory can be accessed by a CPU directly via physical address, but there is usually an intermediate component between CPU and memory called memory management unit (MMU). It is a specialized component that arranges physical memory in a set of pages, addressed by a set of virtual addresses which are then provided to the CPU for memory operations. It makes the memory access substantially faster and more efficient. It also provides the ability for a system to assign a specific process with section of memory and to control it, such that sections from different processes do not interleave. The functionality of the MMU is controlled by a

system software, it can be part of an operating system or a dedicated task for this purpose. The MMU is a first step towards spatial isolation in modern computer systems. A memory protection unit (MPU) is component that ensures no unauthorized memory access is allowed during execution. System software provides additional level of isolation, it controls hardware to provide higher levels of software, namely application software, required functionality. An operating system is a set of system functions that provide the ability for multiple applications to share the same hardware platform. A standard operating system ensures that hardware can be used by several applications simultaneously. Each application is divided tasks which are then organized in processes, the isolation performs on a process level. More specific solutions like microkernel based operating systems or hypervisors provide fully isolated partitions which could host whole partitions and even operating systems themselves. In general, even most complex architectures provide the ability to host multiple applications and provide spatial isolation. The advances in the system software technology (e.g., virtualization) enabled highly advanced control over hardware mechanisms and ensured properties like spatial isolation.

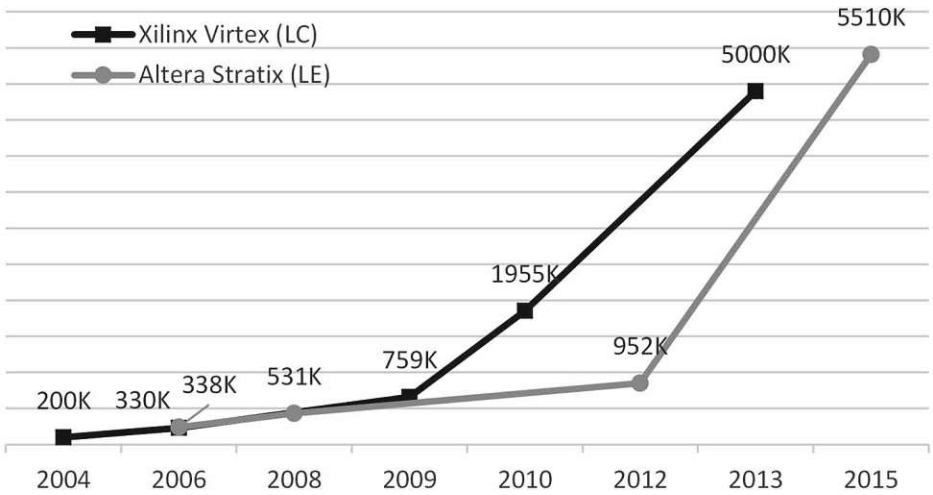
The temporal isolation is a multilayer problem which spreads from hardware architecture to design and execution of both system and application software. The development of hardware architectures was driven by an ever-growing hunger for ability to perform more tasks in less time. As the frequency of processors grew faster and memories grew bigger it was essential that data latency between these two are reduced to a minimum. This is why a number of mechanisms were introduced to accelerate date transfer between CPU and memory. A typical example is the cache memory. It is a high-speed memory built in between the CPU and memory intended as an intermediate storage for a date or instructions that might be used more frequently. Some parts of a program execute more than others and there are mechanisms that predict these repetitions and preloaded them in the cache memory thus allowing the CPU to access it faster. The same technique makes it highly difficult to predict the timing constraints for a specific task as they are implemented on the instruction level. The execution of instructions in a CPU is measurable and the execution time of each task in the CPU can be calculated. The memory access latency can also be determined a priory. The caching mechanisms on the other hand, are based on a series of proprietary algorithms with little or no reference to timing behavior. Thus, making the system unpredictable. In order to ensure temporal isolation tasks or processes need to be evaluated at execution time. This process is called static analysis, and its goal is to determine a worst-case execution time (WCET) of each task. It represents a maximum time that a task can take to execute. With this value a scheduling algorithm determines an execution sequence based on specific constraints, such that any interferences between tasks are avoided. Non-deterministic components (i.e., cache) significantly reduce the ability to calculate WCET for any task. A partial solution for this problem is an overcompensation, by providing more time for execution of a task than initially predicted. However, if the architecture scales up from a single-core architecture to a multi-core architecture the ability to provide temporal guarantees drops even lower. The multi-core architectures use virtual threads across cores to ensure maximal performance. They are also implemented with multi-level cache memories, with each core heaving an exclusive L1 cache, while the L2 cache memory is shared among processors. This increases the level of non-determinism immensely. This is perfectly understandable as the multi-core architectures were not built for the purpose of temporal isolation of mixed-criticality applications.

2.2 Hybrid System-on-a-Chip

FPGAs and hybrid SoC provide an ability to design a domain and application independent architectures which can be configured according to the needs of a specific use case. The process can be integrated within the design process of the whole system, thus having application specific integration already included in the architecture by design (El Salloum, Elshuber, Höftberger, Isakovic, & Wasicek, 2013). Originally FPGAs were used as a prototyping platform where the designers would test hardware components before going for a more permanent ASIC solution. As the technology advanced the FPGAs became more reliable and are more frequently used in different applications, especially where hardware reconfiguration is a system requirement. It ensures longevity of the platform and flexibility if additional applications or functions are necessary. It is an economical approach that which allows designers to define certain properties of the system already in hardware, which increases overall efficiency of the system. In the last decade, FPGAs improved immensely by means of energy efficiency, capacity and pricing. They reached the point where they can be used as a viable replacement for standardized, generic components in hardware platform design. They are heavily utilized in applications that require hardware flexibility and dynamicity (e.g., deep neural learning, satellite on-board computers). Figure 4 illustrates a surge in the progress of FPGA technology in recent years.

Some of the most relevant innovations surrounding FPGAs that have been commercialized in recent years are hybrid SoC architectures, dynamic partial reconfiguration of FPGA fabric and high-level synthesis (HLS). Although all three concepts are not a novelty as such, they came under focus as of recent as the necessary technology became more accessible. All three concepts could be highly important for mixed-critical integration and integration as properties in CPSs in general. The high-level synthesis is a digital design method that creates digital hardware based on abstract behavioral description (McFarland, Parker, & Camposano, 1990). A system described using abstract programming language (i.e., C++) is automatically translated into functional hardware components that perform the same task. Having a

Figure 4. The advancement of FPGA technology in terms of fundamental logic blocks
Altera, 2016; Xilinx, 2016.



hardware implementation of a function can be manifold beneficial (e.g., performance, energy consumption) and having an automated way to implement it is a great advantage.

The dynamic partial reconfiguration of FPGAs allows the system to change the functionality while the system is still operational. A standard way of configuring or programming FPGAs is considered a full re-configuration. The idea behind the concept of partial reconfiguration is analogue to the use of dynamically linked libraries (DLLs) in software applications, a component or a function is added or removed on the fly (Horta, Lockwood, Taylor, & Parlour, 2002). The use partial reconfiguration is suitable for the applications built from multiple configurations which are not active at the same time. It is the way to optimize the resource management of the FPGA. It can be used to increase the fault tolerance of a system in general, by allowing the system to reconfigure certain parts in case of a fault or an error.

The hybrid SoC is a combination of two different hardware design approaches, a multi-core general purpose computing architecture and an FPGA device on a single chip. It represents a synergy of two concepts that operate on different sides of the application spectrum. The general purpose multi-core architectures were designed with the primary goal of increasing performance in general purpose applications (e.g., personal computers, server computers, multimedia devices). The FPGAs traditionally utilized as prototyping devices or used in particular applications where reconfigurable custom hardware design enhances overall system performance, ensures flexibility and ability to adapt the system if necessary (e.g., network routers and switches). The hybrid SoC ensures both by intelligently connecting both architectures in a single device. It allows both devices to operate in isolation while at the same time it is possible to extend one with the other.

The forerunners in hybrid SoCs technology are Xilinx with Zynq 7000 series devices (Xilinx, 2016), Altera (as of recent Intel) with Cyclone and Arria devices (Altera, 2016), and MicroSemi with SmartFusion2 devices (Microsemi, 2016). The devices share the same basic concept and all use ARM architectures, the differences are mainly in the FPGA devices and design tools. The hard-coded processor units are based on ARM Cortex series devices (i.e., A9, M3).

The interface between ARM cores and FPGAs is established over AXI bus bridges that provide full integration of devices in a single system, or partial integration as resources dedicated to each device can be channeled directly to the other. They also allow mutual control between the devices, the FPGA can be re-configured using the HPS and HPS can be initialized from the FPGA side. This allows more flexibility and better fault tolerance capabilities compared to a single device.

The hybrid SoC platforms, alone as a COTS device, provide a favorable environment for hosting multiple applications or applications with mixed levels of criticality. However, the full potential of a hybrid SoC platform lies in its ability to serve as an excellent foundation for secondary architectures. These can be modeled by a specific goal (e.g., deterministic and real-time architecture), or they could be designed for a specific domain (i.e., an ECU for the automotive industry), or used to build a specific application. The structure of the architecture is respectfully complex, thus making the development process potentially error prone, but with the improved tool chain interoperability it can be mitigated. As stated above integration capabilities between design tools are equally as important as the integration between system components themselves.

The hybrid SoCs do not solve the above stated problems by default, it actually creates an environment where some of the issues could be resolved more effectively. It is a sandbox problem with the basic premises for mixed-critical applications satisfied. The performance of hybrid SoC platforms varies, but it it's safe to say that combined performance capabilities are above the average, if we take into account that an average CPU frequency in embedded systems is still below 1 GHz (UBM tech., 2013). The capacity

of FPGAs has been increased by multiple factors in the last five years (see Figure 4), this allows more functions to be hosted on FPGAs. There are a few basic approaches for implementing applications on hybrid SoC platforms:

- A software application hosted exclusively on the hard processing subsystem (e.g., ARM dual-core). It provides good performance characteristics, but it basically has the same characteristics of standard multi-core platforms.
- An application hosted on the hard processing subsystem, with parts of the application implemented on FPGA fabric as a type of an accelerator. This approach is application dependent and could only benefit specific cases. However, it might give a significant performance boost.
- An application implemented in hardware with support of the hard processing system. It is a typical use of FPGA, and it is favorable only in certain cases.
- Use of heterogeneous approach with both HPS and FPGA active. The FPGAs could host multiple generic soft-core based components, dedicated to a specific task or application. It provides multiple layers of separation:
  - **Platform Level:** HPS and FPGA
  - **Core Level:** Multiple cores on HPS, or FPGA, and
  - **Process or Task Level:** Multiple tasks on a single core.

This approach provides multiple benefits with the ability to separate applications physically as well as virtually. It enables multiple instances of the same application, thus providing fault-tolerance. The soft-cores could be heterogeneous depending on a type of the function they are implementing (e.g., for signal processing a digital signal processor is a more effective mechanism then a standard CPU). Although, the soft-cores offer lower performance capabilities with the higher FPGA capacity this could be compensated with high number of components. The components could be individual or connected via on-chip interconnect in a network-on-chip or a bus topography.

This article provides an overview of the approach an overlaying architecture on top of a hybrid SoC platform designed to withhold specific standards and ensure the properties necessary for the mixed-critical integration. A similar approach is described in the next sections of this article. Section 3 describes an architecture that integrates both devices in a reliable and deterministic fashion. It shows how the hybrid SoCs can respond to the challenges of cross-application and cross-domain integration.

2.3 A Time-Triggered Many-Core Architecture

Two main requirements for mixed-critical integration are spatial and temporal isolation. A way to separate applications based on resources and ensure the temporal integrity between them. The commercial solutions are driven with performance capabilities, not intended for safety critical applications or applications with mixed criticality. A time-triggered many-core architecture ACROSS MPSoC was proposed to overcome limitations of the commercial off the shelf solutions (El Salloum, et. al., 2013). It uses time-triggered communication technology and FPGA programmable logic to implement a many-core architecture capable of executing multiple applications, with different criticality levels and with different domain of operation. It is using a service oriented approach with trilateral service hierarchy: core services, optional services and application specific services.

The ACROSS MPSoC is a many-core system with eight soft-coded cores organized in self sustainable components. Each component was a fully functional computer system. The components were assigned particular roles to accommodate service oriented approach. Four components were reserved for core services and optional system services, and the remaining four as application components. Application components can be interfaced with peripheral devices directly or through system components. The backbone of the SoC is a time-trigger network-on-a-chip (TTNoC). A message based communication system controlled by a notion of time. Each component was connected to the TTNoC via the generic interface called trusted interface sub-system (TISS). A property of time-triggered communication is that all members of a network share the same notion of time, called global time. The messages are transmitted between members using a fixed time schedule and it is possible to determine full causality of events between components. This enables system to behave in a fully deterministic fashion. The architecture implements both spatial and temporal isolation, the properties needed for the mixed-critical integration.

A component includes a single-core Nios 2 processor, tied with a physically independent memory unit. The only connection with the rest of the system was trough the deterministic NoC. Thus, each component operated fully independent if not otherwise configured. Each component is defined by a specific set of IO connections, depending on a component's function within the system.

The service oriented approach allows use of the architecture for multiple types of applications and multiple industrial domains. The core services provide basic properties which ensure that the system can be used from a safety critical to a non-critical application. The additional two levels of services, mold the architecture towards a specific domain or application, as additional services can be introduced and integrated into the hierarchy as they are sitting on top of the cores which are ensuring the integrity of the basic properties. The core services would be: configuration, TTNoC communication, execution control, global time, and diagnostics. The optional services are services that provide system level functions, but they are not necessary for the operation of the MPSoC (i.e., operating system, security, IO communication, monitoring, etc.) Application specific services allow the end user to integrate application specific functions into the system. Examples would be: a video service or human interaction interface (HMI) service. A service management layer is missing from the original design. Full service oriented architectures provide a service management services such that users can subscribe or unsubscribe from a specific service. In this case a static configuration of services is necessary. The whole architecture is based on the concept of static configuration at this instance. A dynamic configuration or reconfiguration is extremely difficult to certify for safety applications. One of the goals of this architecture in addition to the feasibility study for mixed criticality integration was evaluation of certifiability of such architecture for safety critical applications.

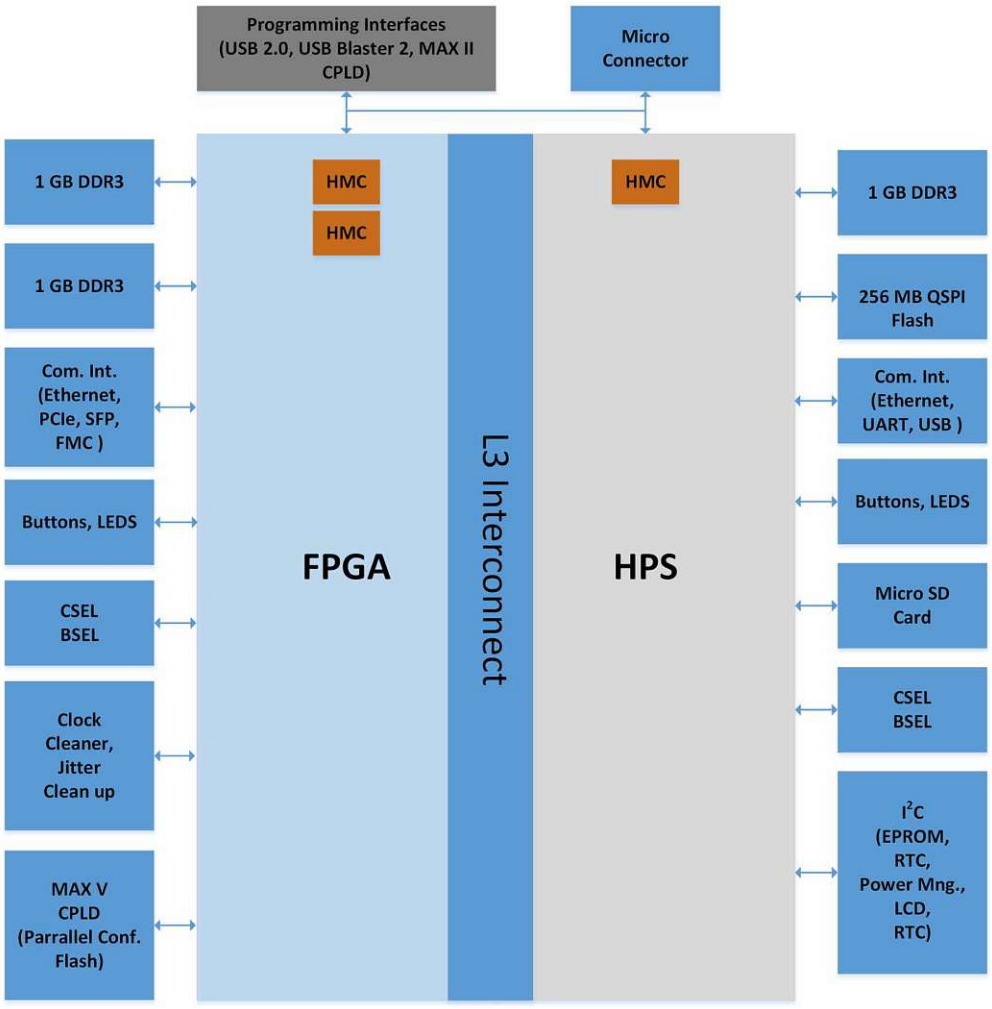
3. A HETEROGENOUS TIME-TRIGGERED ARCHITECTURE ON A HYBRID SOC PLATFORM

The ACROSS MPSoC provided a concrete solution for the problem of mixed-integration. Some of the ideas were not explored, although they seemed fully reachable. The platform offers the ability to increase or decrease the number of components as the TTNoC infrastructure is fully flexible it was able to connect up to 250 components in the current implementation. The TTNoC allows use of heterogeneous components, however, this hasn't been applied due to limitations of the underlying platform. Also, the ability to scale up and implement higher powered components. The hybrid SoC architectures allow more

space for these experiments. In the following section a heterogeneous time-triggered architecture implemented on hybrid SoC (HTTSoC) is described (Isakovic & Grosu, 2016). It is based on the ACROSS architecture with an extension to accommodate a hard-coded ARM processor component. This approach also slightly deviates from the original static service-oriented approach, and explores the possibility of modular architecture with generic component templates. The architecture is organized in three parts: the TTNoC, a cluster soft-core components, and a single hard-coded component. The tunic has the same core from the ACROSS MPSoC, with a different topology and an additional TISS interface to connect the hard-coded component. A component is a fully functional computing device similar to a microcontroller, it is a system within a system, and they were marked as μComponent. The fundamental elements of the communication infrastructure the TTNoC, TISS and a component called trusted resource manager (TRM) are essential for the structure and organization of the architecture.

The implementation uses Altera Arria V SoC as an underlying platform. Figure 5 depicts a block diagram of Arria V SoC. It contains an ARM Cortex A9 dual-core processor described as a hard proces-

Figure 5. A block diagram of Arria V SoC



sor system (HPS) and an Arria V FPGA device. Each processor core runs with a frequency of up to 1 GHz, 32 KB L1 cache memory, and 512 KB of L2 cache memory. In addition, the SoC is equipped with three blocks of 1 GB DDR3 memory and a number of other peripheral memories and IO interfaces that can be used by both devices. The SoC uses AMBA Network Interconnect (NIC-301), marked as an L3 Interconnect, as an on-chip communication backbone between FPGA and HPS.

A core of the communication infrastructure is the TTNoC, a mesh of an arbitrary number of multidirectional switches, called fragment switches interconnected in a specific topology. A fragment switch is a module with four bi-directional channels that relay data either to another fragment switch or to a µComponent interface. Topology of the TTNOC can vary, it is not necessary that each channel is occupied, but the most effective layout is of course a mesh where each channel is closed. It increases routing capabilities and increases maximum throughput and efficiency of the network. The size of the network is bounded by clock distribution limitations of an underlying platform. Essentially, there are two types of data transmitted over TTNoC, time-triggered messages and signals for event-triggering and maintenance of global time.

A next essential element in the communication infrastructure of the architecture is a trusted interface subsystem (TISS). It is trusted interface between TTNoC and components. It is an information gateway where time-triggered message is sent and received transparently according to a predetermined schedule. It has a private memory for storing data and configuration. This is why it serves at the same time it represents a safeguard for the resilience of the network. A TISS is interfaced with the µComponent from one side and with a fragment switch from another. The µComponent interface contains a dual-ported memory for message communication, which is considered to be a part of the µComponent, a set of interrupt signals for event triggering and set of status and synchronization registers.

A trusted-resource manager (TRM) is a network management which is implemented on one of the components connected to the network. It serves as a configuration manager for all TISSs, delivering configuration information for communication schedule and global time. It is the only component allowed to configure TISS located communication schedules and global time base references. It is the only component fully ported from the ACROSS architecture, where the other components are implemented in a generic fashion to explore a modular approach for the design and organization of the MPSoC. The TRM is equipped with a global time generator which is used as the reference tick generator for global time base.

These three subsystems ensure basic functionality of the TTNoC, they are further interfaced with µComponents. In ACROSS MPSoC the µComponents were designed to conform the service-oriented approach, each component was equipped with a set of functionalities to provide a specific service or set of services. The architecture presented in (Isakovic & Grosu, 2016) describes only two types of components HPS Component and a standard soft-core µComponent. The µComponents are implemented using a generic template which can be adapted during the design to specific services. The idea is to create a stable base which or sandbox architecture which can be extended by an end user with a set of premade or custom designed apps. The concept includes integration of design tools such that a custom high-level design would be molded on the TTNoC based SoC.

The second type of component is an HPS component. It uses part of the HPS of the hybrid SoC with a single ARM core and DDR3 SDRAM. The HPS component is interfaced with a corresponding TISS and required memory interfaces located in FPGA. The component itself is also hybrid as the part of the component lies on the HPS and the other part is implemented on the FPGA. The platform allows direct coupling of the hardware components implemented in FPGA with the HPS over L3 interconnect bus bridges. To establish a full functional connection the interface between HPS and TISS requires two L3

Figure 6. ACROSS MPSoC architecture

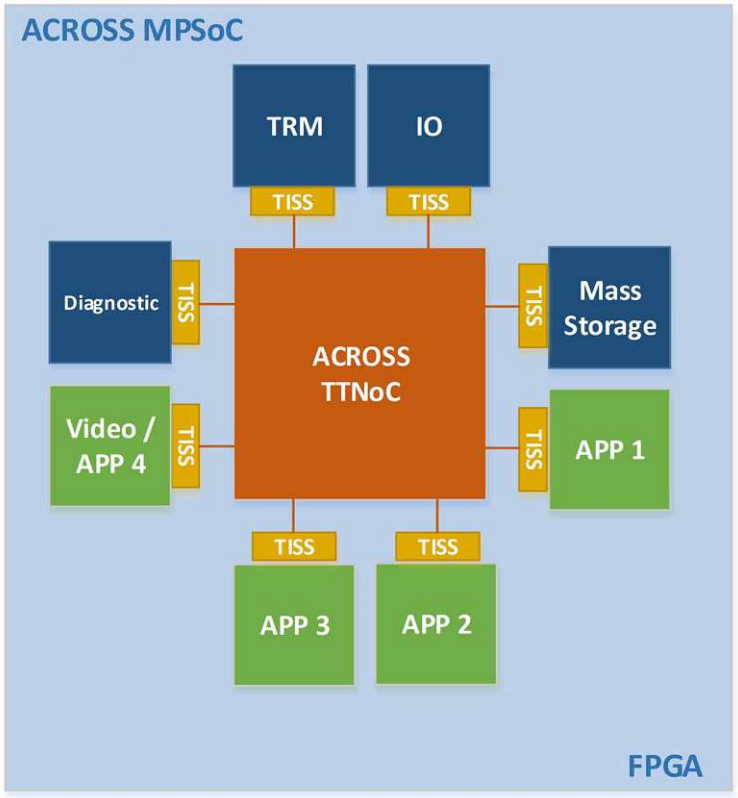


Figure 7. Service hierarchy in ACROSS MPSoC

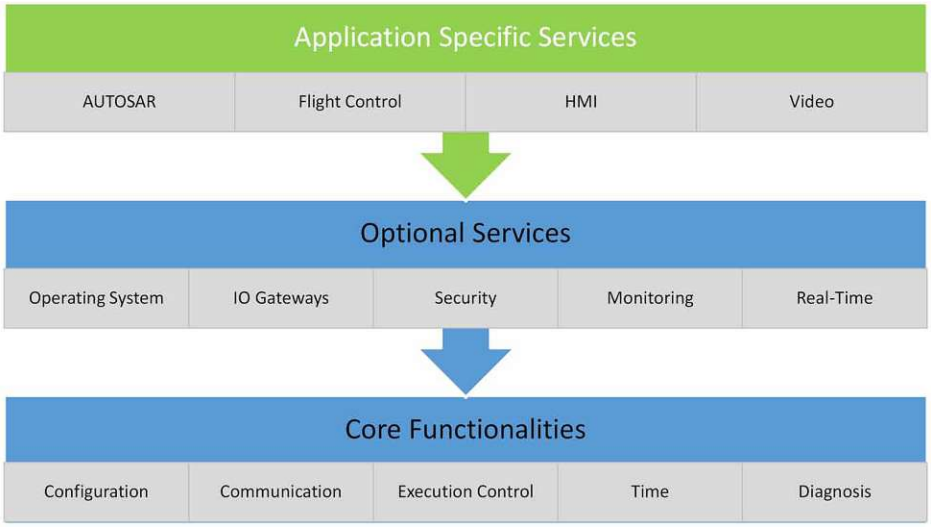
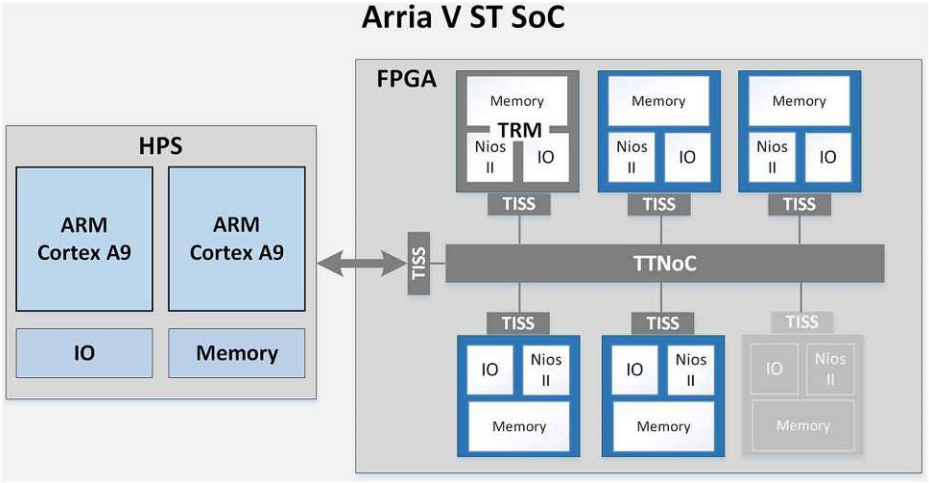
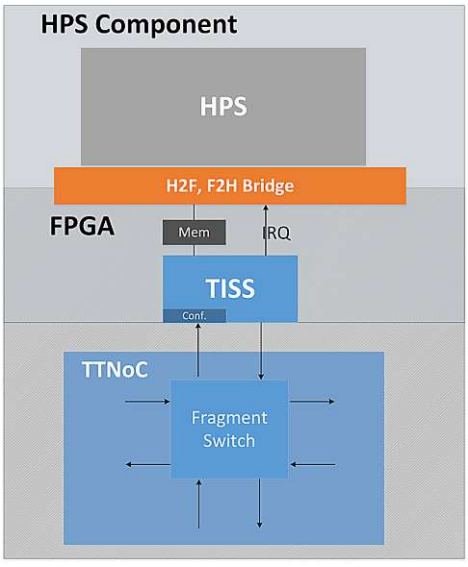


Figure 8. Heterogeneous time-triggered architecture on a hybrid SoC platform
Isakovic & Grosu, 2016.



bus bridges. Two interfaces used in the in this case are light HPS-to-FPGA, and FPGA-to-HPS bridge (see Figure 9). One bridge is used for interfacing dual-ported memory and the other for the routing of interrupt signals. In the initial setup, only one ARM core was used, however, both cores could access the interface if needed. The component is distributed on two devices within a single chip, but it is represented as a single component in design tools. The HPS component still uses default memory hierarchy what makes it less applicable in safety-critical cases. However, it is a perfect solution for the applications without safety constraints that require higher performance capabilities. The dual-core processor can be

Figure 9. HPS micro-component
Isakovic & Grosu, 2016.



run in a symmetric multiprocessing mode or in an asymmetric multiprocessing mode. This provides additional width in the design of applications where on core can be used as a work core and the other as a diagnostic or monitoring unit. The HPS component confirms the ability of TTNoC to integrate heterogeneous components. This means that FPGA can host use case specific accelerators which can be integrated in the architecture instead or alongside Nios 2 components. This way application can utilize modular approach to build a application specific architecture from a generic set of building blocks.

The presented architecture represents a corner stone for the idea of a modular hardware architecture which can be molded on a specific application using a set of high level tools. This way utilization of system resources, power consumption, performance is optimized during the design phase. The required set of tools for this objective is not included in the original platform set of tools. The lack of tool integration presents a major challenge and requires significant amount of efforts to fit the given application on the platform. The goal is to create a configuration tool which maps application requirements on to hardware architecture. This would increase usability and scalability of the presented architecture. The tools would cover hardware requirements, system software requirements and system properties (e.g., safety or security). The original architecture was created with a set of basic tools for configuration of the architecture. To explore full potential of the platform a more powerful set of tools is required which can bridge the manual tasks in the design process, from hardware to an application.

In the case of ACROSS MPSoC each component is considered fault containment unit and the MPSoC as a whole an error containment unit. The global time and synchronization mechanism of TTNoC provide an excellent foundation for component replication. The heterogeneous approach adds another layer of fault and error containment. Although the HPS system is integrated into the TTNoC architecture, it is still a fully independent unit, and the faults in FPGA cannot resonate to the HPS and vice versa. However, the system provides a set of interfaces that enable bidirectional reprogram or reconfiguration. Both devices can be used to monitor and replicate the functionality of the other. The component and the chip represent two levels of fault tolerance in the solution with only FPGA device. In case of hybrid SoC platform there is an additional level between two devices. The above-mentioned tools would be able to integrate a fined grained fault tolerance mechanisms based on this basis.

Tested architecture implementation has four µComponents with a rudimentary design, and a HPS component. A huge factor in the selection hardware platform during application design process is the cost. One of the goals is to explore usability of the presented architecture on a larger scale. The absence of tools that bridge the gaps between hardware design, architecture configuration and application design was mentioned as one of the major challenges. However, it is not the only one in this case as the presented hybrid SoC is still relatively expensive. The modular approach examines the minimal set of components necessary for this architecture to operate. The current design uses about 12% of the FPGA logic elements, 22% of block memory bits and 12% of pins. This platform is capable of hosting a much larger architecture. Also, the modular approach allows us to implement this platform even on the low scale platform, making it more accessible. This also means that the architecture can be scaled down on for low power and low cost applications (e.g., industrial IoT). Deterministic computing in IoT is a major challenge and the time-triggered nature of the presented architecture provides a perfect basis for integration with time-sensitive networks (TSN).

The modular approach in a design process of HTTSoC is a way to explore the optimal design process in CPS where an application can be modeled from the physical environment and computer hardware, system software and application software. The goal of mixed-critical integration is optimal performance, energy consumption and functionality. The FPGA was created with the purpose of having customized

hardware for specific applications, however design process is more challenging than the process of writing software for a generic off-the-shelf computer architecture. The described architecture aspires towards this objective of hardware that can be programmed using established programming techniques and still be able to optimize performance and other system properties according to the requirements of a specific application.

The tool integration is still behind this objective and interoperability needs to be improved in order to allow the integration seamless hardware, software and user tasks and functions.

4. DISCUSSION

This article reflects on the problem of mixed-criticality integration in the CPS, the challenges that are met upon the development of each part of a CPS. From the historical overview of the computer systems and cyber-physical systems in Section 1.2 can be concluded that the integration question dominated throughout all stages of development. Integration is the act or process or an instance of integrating as incorporation as equals into society or an organization of individuals of different groups (as races) (Merriam-Webster, 2016). In the scope of CPS the integration depending on the segment where it is applied e.g., integration between physical system and computer system, integration between hardware and software, or integration between different tasks of an application, an integration of all these segments together. The mixed-criticality integration refers to the incorporation of functional units with different priority levels in relation to a specific system property (e.g., safety, security, real-time). A safety critical system must ensure that all parts of the system are safety proficient, what can be a major overcompensation and extremely inefficient (El Salloum, et. al., 2013). The consumer electronics value user experience as a top priority and this is why the focus is strongly oriented on performance and usability. An average personal computer really reaches the maximum of its abilities; however, it is evaluated on these extremes. The idea of mixed-critical integration also involves partial validation or certification of the system, meaning that only individual components need to be verified for a specific property and their level of criticality.

The performance is a challenge that arises from the current switch in processor technology, in particular single-core processors era is ending and manufacturers are slowly abandoning the production of single-core processors in favor of multi-core CPUs. However, multi-core processors provide insufficient support for safety critical or mixed-critical applications. A TTNoC architecture implemented on a hybrid SoC platform combines hard-coded CPU with a cluster of μComponents based on soft-core CPUs located on an FPGA device in a deterministic fashion without major loss in performance. Moreover, it creates more space for optimization as better utilization of resources. New generations of FPGA devices provide are able to implement extremely fast custom hardware solutions, but also soft-coded micro-controllers more than capable of sustaining state-of-the-art applications. If considered that the processor frequency of embedded architectures which are mostly a subset of cyber-physical systems are averaging around 500 MHz (UBM tech., 2013), it is evident that current hybrid SoC are capable of providing the same service. The next generation of hybrid devices are increasing their performance between 50%-100% (Altera, 2016), it can be concluded that the future platforms would be more than capable of leading mixed-criticality migration.

A system optimization considers multiple measures to create more efficient and stable system. In the concept of the HTTSoC optimization is considered in multiple steps:

- Considering a modular approach as a way to offer sandbox architecture which can be molded on a specific application starting from hardware.
- An integration of design tools, such that mixed-criticality is supported by default and designs optimized transparently.
- Each component can be adapted or replaced with a more effective hardware solution. As the TTNoC generic interfaces can be easily coupled with heterogeneous components.

Having a fully deterministic hardware architecture is currently unreachable, but their numbers of ways to ensure that applications perform deterministically. The communication infrastructure can be deterministic and the TTNoC used in HTTSoC is an excellent example. This means that each information that enters the interface of TTNoC is transmitted in a deterministic fashion, even if the component behind is not. The absence of information is also an information in deterministic networks and can be used to detect faults. As am overcompensation is the main method to achieve determinism on a CPU level, having number of smaller but independent units can achieve the same and even better effect.

The classification of systems is a problem beyond hardware and must be addressed on a larger scale. It is a crucial part of an overall system optimization and a seamless tool interoperability. Having a distributed network of computers with low latency communication on a single platform enables easier segregation of applications according their corresponding sets of functional properties. It is a process that starts with collecting requirements and stretches throughout the design process of each system.

Having an accurate image of a system, *a model*, is pivotal in the design process of CPS. There are examples like the architecture described in (Schuster, Meyer, Buchty, Fossati, & Berekovic, 2014) where a hardware architecture can be modeled in a virtual environment and almost fully evaluated. Such architecture is portable to FPGA design and can be implemented on hybrid SoC platforms. Although, it was not explored the idea of creating a virtual model of HTTSoC has been considered as a part of the tool chain. It would serve as a prototyping platform, the task that was exclusively reserved for early FPGA devices.

The HTTSoC architecture, together with ACROSS MPSoC, fully support the concept of mixed-criticality integration and explores other properties which would improve the design and implementation process of CPS in general. It strongly enforces spatial and temporal isolation which represent a cornerstone for mixed-criticality integration and safety applications.

4.1 Use Case Analysis

The following section provides a short analysis of applications where architecture i.e., HTTSoC might be an advantage. The analysis is concentrating on automotive domain as a one of the largest groups of CPSs. The automotive industry is an industrial with the highest number of manufactured units annually, approximately 90 million units in 2015 (OICA, 2015). The number of ECUs in a car averages between 50 and 100 ECUs depending on the size and a class of the car (Johnson, Gannamaraju, & Fischmeister, 2015). The trends show that a number of functions in a car increases with every generation. The applications like ADAS, car-2-car communication or car-2-OEM communication are becoming a part of the standard car package. The complexity of these new function requires a significant increase in computing power and increase in a number of ECUs. Without merging functionalities, the costs in energy, weight, and fuel economics would surpass benefits. The architectures like HTTSoC and ACROSS MPSoC provide a solution for integrating multiple function new functions and optimize existing functions in a vehicle. The approach not only provides a platform for integration of mixed-critical integration and implementation of

safety-critical application, but also offers a way to reduce power consumption and weight of a car. This could significantly improve fuel economy in a car. A number of communication channels and cables in a car puts significant overhead on the energy consumption and weight. Use of HTTSoC or ACROSS MPSoC enables virtualization of communication channels, like CAN lines that reduces the number of physical devices in the car (Wasicek, Höftberger, Elshuber, Isakovic, & Fleck, 2014).

Another major challenge in a car industry is security, the performance of a car and other safety related aspects are tightly connected with other non-essential systems in a vehicle. Also, it is crucial that only verified replacement parts can be installed in a car. Use of TTNoC reduces number of physical connections of the system, this a significant feature as even non-essential channels can be source of security vulnerabilities. *Security* as a property is supported by design in HTTSoC and ACROSS and with few additional features can be transformed in a secure platform, fully transparent to the end user (Isakovic & Wasicek, 2013).

5. FUTURE DIRECTIONS

As stated earlier in the text implementation of hardware without support of software and design tools leaves more space for design faults. Full tool interoperability that captures a design process of a CPS from a requirement phase to deployment is a future objective. The tool chain for the development of hardware and software in the described architectures is a rather complex structure. This could be simplified with a couple of bridges in the configuration scheme.

Another goal is to fully use heterogeneous design capabilities and integrate other types of cores, a typical example is Leon 3 SPARC processor from Geisler (Geisler, 2016). It has a fault-tolerant configuration and it is highly utilized in the space domain. Integrating this process would open the door for the architecture in space related applications.

To increase the spectrum of applications the architecture needs to be expanded on different hybrid SoC platforms, e.g., Xilinx Zynq. It is a more common platform, with the capability of high level synthesis, which can be used to integrate the design process better and to implement software functions as TTNoC components.

The next generation of hybrid SoCs provides ability to perform a partial dynamic reconfiguration of FPGA fabric. This can be used as a tool to increase resilience of a system, or to enhance the system during runtime. The goal is to explore these capabilities within the presented architecture and provide tool integration for the application development process.

6. CONCLUSION

The number of functions in modern CPS (e.g., motor vehicles) is increasing with each new generation and the way to accommodate the necessary increase in computing power is to merge multiple applications on a single power and reduce the physical footprint of computer hardware in a system. To accomplish this objective a number of obstacles need to be overcome. The performance of the hardware architecture needs to match the requirements of the increasing number of applications and the complexity of applications. The hardware ensures deterministic behavior in order to ensure spatial and temporal isolations

of individual applications. The shift from single-core to multi-core architectures created a vacuum in for safety critical systems, as COTS multi-core architectures provide insufficient guarantees for safety critical applications. The safety regulations impose the rule that a system must ensure the safety integrity level of the application with highest safety integrity level. For multi-core architectures with symmetric multiprocessing this increases problem of mixed-criticality integration even further. The qualification and classification of applications according to their functional requirements is necessary to avoid over-compensations during design and implementation process.

The HTTSoC and ACROSS MPSoC provide a viable solution for mixed-criticality integration and a number of ways to face the above stated challenges head on. The introduction of hybrid SoC platforms provided a sandbox environment for the design of hardware architectures that match COTS hardware in terms of performance and outran them in terms of efficiency. It solves some fundamental problems, however there are still more detailed challenges, like tool integration and modular design process that would ensure the full sandbox experience for users and still preserve fundamental properties necessary for mixed-critical and safety-critical applications.

ACKNOWLEDGMENT

The research presented in the article is a product of two major research initiatives from the ARTEMIS Joint Undertaking (JU), in particular projects ACROSS (N° 2009-1-100208) and EMC2 (N° 621429). Also, this work has been supported by the company TTTech a long-term partner of Vienna University of Technology.

REFERENCES

Ampère, A. M. (1838). *Essai sur la philosophie des sciences*. Paris: Bachelier.

Asada, H., Kanade, T., & Takeyama, I. (1983). Control of a Direct-Drive Arm. *Journal of Dynamic Systems, Measurement, and Control*, *105*(3), 136–142. doi:10.1115/1.3140645

BBC. (24. 10 2012). *BBC Ceefax, the world's first teletext service, has completed its final broadcast after 38 years on air.* Retrieved from http://www.bbc.com/news/uk-20032882

Blomstedt, F. (2014). The arrowhead approach for SOA application development and documentation. In *Proceedings of the IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society,* Dallas (pp. 2631-2637). IEEE.

Bowen, J., & Stavridou, V. (1993). Safety-critical systems, formal methods and standards. *Software Engineering Journal*, *8*(4), 189–209. doi:10.1049/sej.1993.0025

Carter, W. S. (1994). The future of programmable logic and its impact on digital system design. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors* (pp. 10-16). Cambridge: IEEE. doi:10.1109/ICCD.1994.331842

DEC. (1967). *The digital small computer handbook.* Digital equipment corp.

El Salloum, C., Elshuber, M., Höftberger, O., Isakovic, H., & Wasicek, A. (2013). The ACROSS MPSoC - A new generation of multi-core processors designed for safety-critical embedded systems. *Microprocess. Microsyst., 37*, 0141-9331.

Geisler. (2016). *Leon 3 SPARC*. Von Geisler: http://www.gaisler.com/index.php/products/processors/leon3?task=view&id=13

Grimwood, J. M. B. C. (1969). Project Gemini; Technology and Operations. Washington, D.C.: G.P.O.

Horta, E. L., Lockwood, J. W., Taylor, D. E., & Parlour, D. (2002). Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In *Proceedings of the Design Automation Conference* (pp. 343-348). IEEE.

IEEE. (20. 11 2016). *Official timeline of 802.11 development*. Retrieved from http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm

Isakovic, H., & Grosu, R. (2016). A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform. In *Proceedings of the 25th International Symposium on Industrial Electronics (ISIE),* Santa Clara (pp. 244-253). IEEE. doi:10.1109/ISIE.2016.7744897

Isakovic, H., & Wasicek, A. (2013). Secure channels in an integrated MPSoC architecture. In *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society,* Vienna (pp. 4488-4493). IEEE. doi:10.1109/IECON.2013.6699858

Johnson, T. T., Gannamaraju, R., & Fischmeister, S. (2015). A Survey of Electrical and Electronic (E/E) Notifications for Motor Vehicles. In *Proceedings of the 24th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, Gothenburg (pp. 1-15).

Johnson, T. T., Gannamaraju, R., & Fischmeister, S. (2015). A SURVEY OF ELECTRICAL AND ELECTRONIC (E/E) NOTIFICATIONS FOR MOTOR VEHICLES. *24th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, Gothenburg (pp. 1-15).

Kernighan, B., & Ritchie, D. (1978). *The C Programming Language* (1st ed.). Prentice-Hall.

Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., & Zainlinger, R. (1989). Distributed fault-tolerant real-time systems: The Mars approach. *IEEE Micro*, *9*(1), 25–40. doi:10.1109/40.16792

McFarland, M. C., Parker, A. C., & Camposano, R. (1990). The high-level synthesis of digital systems. In *Proceedings of the IEEE* (pp. 301-318). IEEE.

Merriam-Webster. (2016). *Merriam-Webster Dictionary*. Von Merriam-Webster: https://www.merriam-webster.com/ abgerufen

NASA. (2016). *Computers in Spaceflight: The NASA Experience*. Retrieved from http://history.nasa.gov/computers/contents.html

NIST. (1995). *An Introduction to Computer Security: The NIST Handbook.*

Nof, S. Y. (1999). *Handbook of Industrial Robotics* (Vol. 1). John Wiley & Sons. doi:10.1002/9780470172506

O'Regan, G. (2012). *A Brief History of Computing*. London: Springer Science & Business Media. doi:10.1007/978-1-4471-2359-0

OICA. (2015). *Production Statistics for Motor Vehicels*. Retrieved from http://www.oica.net/category/production-statistics/2014-statistics/

Ross, D. T. (1978). Origins of the APT language for automatically programmed tools. In R. L. Wexelblat (Ed.), *In History of programming languages I* (pp. 279–338). New York: ACM. doi:10.1145/960118.808374

Rushby, J. (1994). Critical System Properties:Survey and Taxonomy. *Reliability Engineering & System Safety*, *43*(2), 189–219. doi:10.1016/0951-8320(94)90065-5

Schuster, T., Meyer, R., Buchty, R., Fossati, L., & Berekovic, M. (2014). SoCRocket - A virtual platform for the European Space Agency's SoC development. In *Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Montpellier (pp. 1-7). IEEE. doi:10.1109/ReCoSoC.2014.6860690

Studer, R., Benjamins, R. V., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, *25*(1-2), 161–197. doi:10.1016/S0169-023X(97)00056-6

The World Bank Group. (2016). *Mobile cellular subscriptions (per 100 people)*. Retrieved from http://data.worldbank.org/indicator/IT.CEL.SETS.P2?end=2003&start=1984&view=chart

Trimberger, S. (2012). *Field-Programmable Gate Array Technology*. Springer, US.

UBM tech. (2013). *Embedded Market Survey.* UBM.

US Census Bureau. (02 2010). *Computer and Internet Use in the United States: 1984 to 2009*. Retrieved from http://www.census.gov/data/tables/time-series/demo/computer-internet/computer-use-1984-2009.html

Vinoski, S. (1993). *Distributed Object Computing with Corba 1.0 Introduction.*

Wasicek, A., Höftberger, O., Elshuber, M., Isakovic, H., & Fleck, A. (2014). Virtual CAN Lines in an Integrated MPSoC Architecture. In *Proceedings of the 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing,* Reno (S. 158-165). IEEE. doi:10.1109/ISORC.2014.34

Wiener, N. (1961). *Cybernetics Or Control and Communication in the Animal and the Machine*. MIT Press. doi:10.1037/13140-000

Wirth, N. (1971). The programming language pascal. *Acta Informatica*, *1*(1), 35–63. doi:10.1007/BF00264291

KEY TERMS AND DEFINITIONS

ASIC: Application specific integrated circuit.
ECU: Electronic Control Unit.
HTTSoC: Heterogeneous Time-Triggered SoC.
Mixed-Criticality: A system with multiple tasks of different property requirements.
MPSoC: Multiple processor System-on-a-chip.
SoC: System-on-a-chip.
TRM: Trusted Resource Manager.
TTNoC: Time-triggered network-on-a-chip.

CHAPTER 6

CPS/IoT Ecosystem: A Platform for Research and Education

CPS/IoT Ecosystem: A platform for research and education

Haris Isakovic[1], Denise Ratasich[1], Christian Hirsch[1], Michael Platzer[1], Bernhard Wally[1], Thomas Rausch[1], Dejan Nickovic[2], Willibald Krenn[2], Gerti Kappel[1], Schahram Dustdar[1], and Radu Grosu[1]

[1] Technische Universität Wien, Austria
[2] Austrian Institute of Technology, Austria
{name.surname}@tuwien.ac.at {name.surname}@ait.ac.at

Abstract. The CPS/IoT Ecosystem project aims to build an IoT infrastructure that will be used as a platform for research and education in multiple disciplines related to CPS and IoT. The main objective is to provide a real-world infrastructure, and allow students and researchers explore its capabilities on actual use cases.

Keywords: Internet-of-Things · Infrastructure · Cyber-Physical System

1 Introduction

We are experiencing a major paradigm shift in terms of computing systems. The ability to collect big data, use it to model physical environments with astonishing precision and use it to improve existing systems is a principal factor behind the upcoming revolution.We are using Cyber-Physical Systems(CPS) to observe and manipulate our physical environment,and the Internet-of-Things (IoT) to transfer and transform this raw data into profitable information.CPS/IoT Ecosystem is a project that materializes this idea. It embodies an infrastructure for IoT integrated together with a set of use cases that represent CPS. It is a joint project of three research institutions Technische Universitẗ Wien (TU Wien), Austrian Institute of Technology (AIT), and Institute for Science and Technology (IST). It serves as a research platform for a variety of related disciplines and as an educational tool for bringing concepts of IoT and CPS closer to the students in a "hands-on" type of an approach.

The preliminary forecasts state that IoT it will continue to grow rapidly in the next ten years. Multiple studies predict a number of new IoT devices to reach 75-100 billion until 2025 [20]. The global network of IoT devices will include both public and private IoT domains, with the ability to share and monetize not only results but also the usage of the infrastructure itself [8] [15]. We will highlight just few important challenges:

Development. Each scope of operation ion CPS and IoT (e.g., cloud, fog/edge, sensor, network) is traditionally observed as a separate discipline.The development methods and tools for each scope have been created accordingly

and they are not necessarily mutually compatible. To build an IoT application the tools for development, testing, and deployment need to be fully inter-operable.

Management. The holistic idea of IoT is to have billions of heterogeneous devices serving millions of different applications connected to Internet. Running these systems requires configuration, deployment, software updates and maintenance etc. Managing these tasks on a system this magnitude is a major challenge and requires enormous amount of effort.

Security. In the world where "every single thing" is connected to the Internet, security represents crucial requirement. Standardized approach to security and related topics, i.e., privacy and trust must be is a major challenge in IoT. Example from data breaches and recent changes in EU regulation regarding handling private data [26] highlights just how important is security in IoT. In the future data will influence policies and indirectly lives of people. Making sure that the data is valid and secure is extremely important.

Power Consumption. Over 75 billion new devices in just under ten years will create a massive overhead on the existing power infrastructure. The current production of electrical energy in the European Union on a yearly basis is around 3000 TWh [6].An average IoT device like the Raspberry Pi 3 consumes up to 5 Wh of electrical energy.If we pessimistically project it onto new IoT devices we get 3240 TWh on a yearly basis just for these devices.

The project will focus on implementing hardware structure of devices with custom build cloud system, fog/edge nodes, and COTS and custom built sensor and actuator devices that will form an infrastructure. It explores development frameworks, tools and mechanisms that will ensure standardized design and help establish functional system between hardware, software, and applications. A major aspect of the project is its educational value in terms of bringing state-of-the-art technology directly into curriculum. A new IoT infrastructure providing means for realistic implementations and applications, it enables students to experience complexity, real-time, security and dependability issues on real-world examples. Beside using the infrastructure in courses, the infrastructure will become central topic for numerous bachelor and master theses.

In this chapter we introduced the motivation behind the project and its core concepts. In Chapter 2 we provide short overview of the related projects. Chapter 3 describes methodology for the project execution and its most essential components. Two use cases implemented in the project are described in Chapter 4. The final chapter concludes the paper and provides future directions for the project.

2 Related Work

IoT represents a super set of multiple disciplines i.e., machine learning, artificial intelligence, real-time systems, embedded systems, high performance computing, web and mobile technologies, networking , enterprise organization, civil engineering and a number of others. Vermessan et. al. define IoT as *"a concept*

and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things/objects to create new applications/services and reach common goals" [28]. In this section we will give a short overview of the relevant research topics with respect to the CPS/IoT Ecosystem project and related research projects within the scope of EU research community.

The heterogeneity of IoT is one of its most prominent features however on the development and run-time level it is often primary source of **interoperability** issues. This is why the interoperability is one of the most researched topics in IoT. A significant number of projects in are working to enable or increase interoperability between existing and new IoT platforms and devices [5, 14, 18].

Second major topic in the scope of IoT research and development is **security**. It is arguably the most difficult challenge in IoT. It is a rather complex topic, as it branches in a numerous subtopics, each pf which is highly complex and demanding on its own. Thus a large variety of projects and research initiatives on the variations of security and security related topics (e.g., encryption, trust, privacy, block chain, user, data and ip protection) [4, 19, 13, 23].

The IoT represents a large heterogeneous system with a enormous variety of applications. Providing generic rules and guidelines allows us to create systems with standardized system properties. However, systems also need to be tailored to each individual application and its requirements. According to Gabriel in [12] *"a system that can be customized, specialized, or extended to provide more specific, more appropriate, or slightly different capabilities"* is called **framework**. A framework allows us to use it for different purposes without having a need to write the code each time from the beginning. Multiple research initiatives are exploring different frameworks for IoT, with different specialization abilities (e.g., security, safety, service-oriented design, social aspect, education and others)[10, 19, 7, 23, 22, 25].

Providing generic rules and guidelines allows us to create systems with standardized system properties. However, systems need to be tailored to specific application requirements. According to Gabriel in [12] *"a system that can be customized, specialized, or extended to provide more specific, more appropriate, or slightly different capabilities"* is called **framework**. It allows us to use it for different purposes without having a need to write the code each time from the beginning. Building such systems is one of the most explored questions in IoT. Multiple research initiatives are exploring different IoT frameworks with different specialization abilities (e.g., security, safety, service-oriented design, social aspect, education and others)[10, 19, 7, 23, 22, 25].

The applications are strongest driver behind IoT revolution. IoT applications are normally spearheaded by commercial subjects and the number of different ways IoT is improving existing systems changes every day. The research aspects with respect to IoT applications focuses on model and framework design, big data, social and economical implications, security and privacy issues, and

cooperation with other fields of science (e.g., biology, medicine, mechanical engineering, geo-engineering, etc.) [2, 9, 18, 1].

3 CPS/IoT Ecosystem Methodology

CPS/IoT Ecosystem is conceived as a heterogeneous structure of hardware devices, and corresponding software components distributed over tree intertwined scopes of operation: cloud, fog/edge, and sensor/actuator nodes. The cloud provides high performance computation and large capacity storage. The fog, also referred as edge, level indicates a network of devices with real-time communication capabilities, and mid-range computational and storage capabilities. The sensor/actuator level serves as a direct interface with physical environment. They posses capabilities of collecting and transforming physical signals using sensors and manipulating the environment via diverse actuators. The CPS/IoT Ecosystem infrastructure is a geographically distributed system. Parts of the infrastructure will be located on multiple sites on a wider area of Vienna,Austria.

CPS/IoT Ecosystem Cloud The cloud system is a general purpose high-performance computing platform located at a server center of TU Wien. It provides services that facilitate handling of big data (e.g., storage, analysis, aggregation). It is an essential part of the infrastructure. In CPS/IoT Ecosystem we are implementing a custom built cloud server. Its purpose is to serve the applications, but also to be used as a research subject. It will be deployed in two parts: a) a general purpose computing platform, and b) specialized computing platform for calculation intensive tasks.

CPS/IoT Ecosystem Fog/Edge Ability to react fast and ensure quality of service (QoS) on a factory floor level or similar plane of execution is implemented in the fog/edge level. It represents a network of computing nodes which are both capable of handling certain significant of data and still ensure service dependability. The fog/edge devices can be in direct connection to the sensor/actuator nodes or as an intermediate gateways for the ultra low energy/performance devices.

CPS/IoT Ecosystem Sensor Device The sensor/actuator nodes are direct interfaces with a physical environment. These devices are limited in computational performance, size and power consumption. They can be deployed individually or in swarms as explained in Section 4.

CPS/IoT Ecosystem Information Model As mentioned above management, development, and security are three major challenges in IoT. Part of the solution for these challenges is a functional IoT information model for the CPS/IoT infrastructure. It will allows us to describe the system from multiple perspectives: hardware platform, services, application, management and communication. The acquired models can be used as templates for application design, code generation, development and operations (DevOps), testing and validation.

COTS vs. Custom Built Hardware CPS/IoT Ecosystems generally comprises a substantial amount of sensor nodes.Buying a lot of sensor hardware can quickly consume an important amount of a project's budget, since commercially available ready to use hardware is usually expensive. Therefore designing and building custom hardware can be an attractive alternative. Designing custom hardware also has the advantage of increased flexibility, since one is not limited by the choice of existing components. The hardware design can be tailored to specific requirements as described in 4.

Technology in CPS/IoT Ecosystem The objective of the CPS/IoT Ecosystem project is to build a technology agnostic IoT infrastructure. Often the IoT is connected to a single framework, communication protocol or cloud environment. This project will provide general purpose cloud environment based on OpenStack[24]. It uses variety of open source and research community frameworks to build middle-ware and application software [17, 10, 27, 11]. In CPS/IoT Ecosystem we are not limited to a single communication protocol, typical IoT communication standards described in [3, 16, 21] will provide a basis for networking and communication standards.

4 Use Cases

4.1 Smart Parking

Smart Parking application provides status information of public or private parking places in a city or garage. Each parking spot is equipped with a sensor or group of sensors capable of detecting objects (cars or similar) on a surface of the parking spot. Data of each sensor is then transmitted to a central application software located on a remote server via Internet connection. The information is further delivered to end user over Web or mobile application. The Smart Parking application is build on the principle of CPS/IoT Ecosystem. The cloud environment serves as mass storage device and a service provider to external users. It collects all parking information from the fog all fog nodes and provides them to external applications (e.g., web site). The edge/fog nodes serve as sensor data aggregation and filtering nodes. The data collected from the sensors is transformed in the application useful information. The parking spot is a virtual concept and can be formed on arbitrary surface with a single or multiple heterogeneous sensors. Figure 1 provides an architectural overview of the Smart Parking application. The middleware backbone of the Smart Parking application is a service-oriented Arrowhead IoT Framework[10]. The Smart Parking application services are distributed over local clouds both on cloud and edge/fog level of operation. Senor nodes are connected via Bluetooth Low Energy (BLE) protocol. Future work on this use case considers adding multiple additional services (e.g., payment, allocation and reservation of spaces), also adding support for multiple sensor types and building vehicle-2-infrastructure interface for autonomous parking. Another feature is the implementation of sensor node simulator which is able to project sensors on the scale of the city and provide simulated data to the

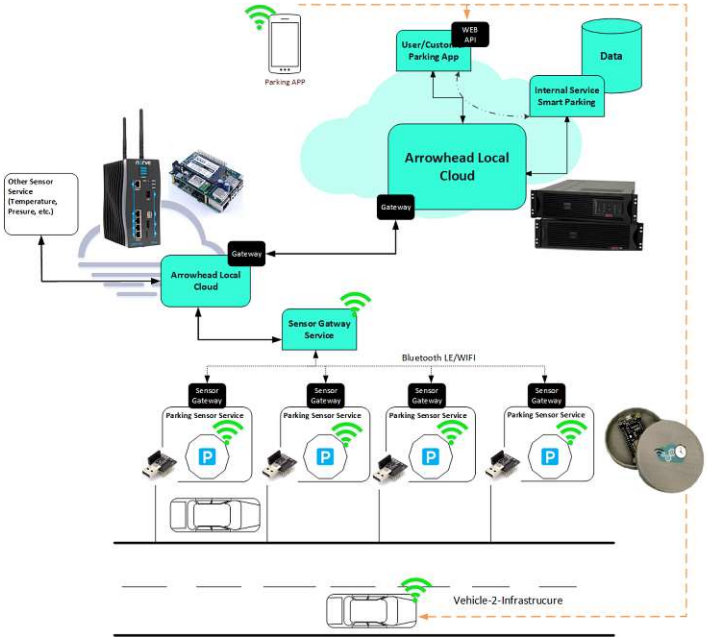


Fig. 1. Smart Parking Architecture Overview

rest of the infrastructure. This allows us to test scaleability and manageability of the application without deploying hardware devices.

4.2 Smart Vineyards

IoT can help to overcome arising problems in the agricultural sector. For example, the increasing labor shortage do to the depopulation of rural areas. This is possible because such infrastructures help farmers to work more efficiently. Examples are disease prediction systems, that warm farmers of arising diseases in certain areas. Farmers can use this information to bring out pesticides only when it is necessary and also only where it is necessary. This reduces the workload of farmers, the costs for pesticides and the negative impact on the environment.

We are building such an infrastructure for vineyards in cooperation with the Vienna University of Natural Resources and Life Sciences (BOKU Wien) as part of the CPS/IoT Ecosystem project. The aim is to bring out several hundreds of swarm nodes that measure the environment. This information is transmitted to the cloud via fog nodes. Later, the information is processed by means of big-data analysis and machine-learning algorithms to learn correlations between diseases and environmental influences to create new and improve existing diseases prediction models.

5 Conclusion

The paper provides a short overview of the CPS/IoT Ecosystem project and its main objectives. IoT has a complex and broad spectrum of topics and CPS/IoT Ecosystem is providing a platform where these topics can be explored and also bring closer to the students. Our goals are to build a physical infrastructure, to ensure data flow and application integration, demonstrate multiple use cases, and open it to other research initiatives for collaboration. The project is in its first stage and will continue to develop, thus our focus in the future will change on research of methods how to improve management, development, security and other properties.

6 Acknowledgment

This work has been conducted within a project that has received funding from the Austrian Government through the Federal Ministry Of Education, Science And Research(BMWFW) in the funding program Hochschulraum-Strukturmittel 2016 (HRSM).

References

1. AfarCloud: Aggregate Farming in the Cloud, https://www.ecsel.eu/projects/afarcloud
2. AGILE: Aircraft 3rd generation MDO for innovative collaboration of hetergeneous teams of experts, https://www.agile-project.eu
3. Al-Sarawi, S., Anbar, M., Alieyan, K., Alzubaidi, M.: Internet of things (iot) communication protocols: Review. In: 2017 8th International Conference on Information Technology (ICIT). pp. 685–690 (May 2017). https://doi.org/10.1109/ICITECH.2017.8079928
4. ARMOUR: Large scale experiment of IoT security and trust, https://www.armour-project.eu/
5. BIGIoT: Bridging the Interoperability Gap of the Internet of Things, http://big-iot.eu
6. BP: Bp statistical review of world energy. bp-statistical-review-of-world-energy-2017-full-report.pdf (June 2017), https://www.bp.com/content/dam/bp/en/corporate/pdf/energy-economics/statistical-review-2017/bp-statistical-review-of-world-energy-2017-full-report.pdf
7. Brain-IoT: A model-Based Framework for dependable sensing and Actuation in Intelligent decentralized IoT systems, http://www.brain-iot.eu
8. Chen, S., Xu, H., Liu, D., Hu, B., Wang, H.: A vision of iot: Applications, challenges, and opportunities with china perspective. IEEE Internet of Things Journal **1**(4), 349–359 (Aug 2014). https://doi.org/10.1109/JIOT.2014.2337336
9. CLOUT: Cloud of Things for empowering the citizen clout in smart cities, http://clout-project.eu/
10. Delsing, J.: IoT Automation: Arrowhead Framework. CRC Press (2017), https://books.google.at/books?id=6mMlDgAAQBAJ

11. FU Berlin: RIOT - The friendly Operating System for the Internet of Things. available at http://riot-os.org, accessed 2018-07-18
12. Gabriel, R.P.: Patterns of Software: Tales from the Software Community. Oxford University Press, Inc., New York, NY, USA (1996)
13. GhostIoT: Safe-Guarding Home IoT Environments with Personalised Real-time Risk Control, https://www.ghost-iot.eu
14. InterIoT: Interoperability of heterogeneus IoT platforms, https://vicinity2020.eu
15. Kyriazis, D., Varvarigou, T., White, D., Rossi, A., Cooper, J.: Sustainable smart city iot applications: Heat and electricity management amp; eco-conscious cruise control for public transportation. In: 2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM). pp. 1–5 (June 2013). https://doi.org/10.1109/WoWMoM.2013.6583500
16. Ngu, A.H., Gutierrez, M., Metsis, V., Nepal, S., Sheng, Q.Z.: Iot middleware: A survey on issues and enabling technologies. IEEE Internet of Things Journal **4**(1), 1–20 (Feb 2017). https://doi.org/10.1109/JIOT.2016.2615180
17. Open Source Robotic Foundation, Inc.: Robot Operating System (ROS). available at http://www.ros.org/, accessed 2018-07-06
18. Productive4.0: Electronics and ICT as enabler for digital industry1 and optimized supply chain management covering the entire product lifecycle, https://productive40.eu
19. RERUM: REliable, Resilient and secUre IoT for sMart city applications, https://ict-rerum.eu/
20. Rose, K., Eldridge, S., Chapin, L.: The internet of things: An overview. ISOC-IoT-Overview-20151221-en.pdf (Feb 2015), https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf
21. Schachinger, D., Kastner, W.: Semantic interface for machine-to-machine communication in building automation. In: 2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS). pp. 1–9 (May 2017). https://doi.org/10.1109/WFCS.2017.7991956
22. Semiotics: Secure Multi-protocol Integration Bridge for the IoT, https://www.semiotics-project.eu
23. SerIoT: Secure and Safe Internet of Thing, https://seriot-project.eu/
24. Shrivastwa, A., Sarat, S., Jackson, K., Bunch, C., Sigler, E., Campbell, T.: OpenStack: Building a Cloud Environment. Packt Publishing (2016)
25. SOCIOTAL: Creating a socially aware citizen-centric Internet of Things, http://sociotal.eu/
26. THE EUROPEAN PARLIAMENT: Regulation (eu) 2016/679 of the european parliament and of the council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679\&from=DE
27. Thingsboard: Thinsboard IoT Platform, https://thingsboard.io/
28. Vermesan, O.: Internet of things : converging technologies for smart environments and integrated ecosystems. River Publishers, Aalborg, Denmark (2013)

CHAPTER 7

QoS for Dynamic Deployment of IoT Service

QoS for Dynamic Deployment of IoT Services

Isakovic Haris*, Luis Lino Ferreira †, Irmin Okic*,
Adam Dukkon*, Zlatan Tucakovic*, Radu Grosu*
Technsiche Universität Wien*, Vienna, Austria
CISTER Research Center†, ISEP Polytechnic Institute of Porto, Porto, Portugal
* name.surname@tuwien.ac.at, † llf@isep.ipp.pt

***Abstract*—This paper introduces RVAF, a runtime verification (RV) extension of the Arrowhead Framework (AF) with container-based service-deployment and runtime-enforcement of a desired quality of service (QoS). AF is a service-oriented middleware architecture for IoT-applications, consisting of a set of core and auxiliary services and systems, respectively. The QoS manager (QoSM) is one AF's most important auxiliary systems, which can be used to guarantee the application's QoS for a wide set of parameters. In RVAF the QoS offered to a particular IoT-application is specified in signal temporal logic, and is continuously monitored by the RVAF-QoSM. In case of an imminent violation, RVAF automatically initiates a container-based reconfiguration, which is ensured to maintain the desired QoS. RVAF is beneficial to large IoT-applications, where the use of continuous-integration and continuous-deployment tools, is not only a recommended practice but also a necessity. Moreover, the use of RVAF is advantageous both during the development of an IoT application, and after its deployment. We describe the architecture of RVAF, provide its formal underpinning, and demonstrate the usefulness of RVAF supported by an industrial IoT application. The main contribution of this work is to show what it takes to incorporate RV concepts into modern SOA frameworks supporting the development of IoT applications.**

I. Introduction

Industrial digital systems (IdS) were originally designed with a mission-oriented focus in mind. As a consequence, IdS were homogeneous pieces of software, most often unable to adapt to the changes occurring during their evolution, such as the IdS' subsequent integration with management, analysis, or financial systems, respectively. In order to alleviate these problems, the design of IdS is now increasingly adopting the IoT paradigm, which is one of the major pillars of the industrial-digitization initiative to optimizing production chains [1].

The arrowhead framework (AF) is a middle-ware enabling the service-oriented approach (SOA) to the development of IoT applications [2]. SOA facilitates the construction of generic services, that are later on integrated in various applications, as application-specific services. SOA thus greatly enhances the reuse of services, and their integration into fine-grained functionalities.

AF includes a set of core services, such as service discovery, orchestration, and authentication, which in turn support the interaction between application services, such as sensor-reading or data-storage services [2]. Each service is a part of a system-of-systems (SoS) and acts as a completely separate software entity. As such, services are developed, operated, and maintained by different groups, organizations, or individuals. Ensuring a seamless integration between development and maintenance on the one side, and the quality of service (QoS) during operation on the other side, are overlapping tasks.

This paper introduces RVAF, a continuous integration and deployment, RV-extension of AF. RVAF supports the continuous service integration/deployment in a container-based fashion, and the runtime enforcement of a desired quality of service (QoS) through adaptation. In RVAF the QoS guaranteed to a particular IoT-application is specified in signal temporal logic (STL), and continuously monitored by the QoS manager (QoSM). As shown in Figure 4, in case of an imminent violation, RVAF automatically initiates a container-based reconfiguration, which is ensured to maintain the desired QoS. RVAF is beneficial to large IoT-applications, where the use of continuous-integration and continuous-deployment tools, is not only a recommended practice but also a necessity. Moreover, RVAF use is not only beneficial after the deployment of an IoT application, but also during its development. In addition to describing the architecture of RVAF and providing its formal underpinning, we give proof of concept of the usefulness of RVAF based on a real-life, industrial scenario.

The rest of the paper is organized as follows. In Section II we introduce AF, followed by related work in QoS on Arrowhead. In Section IV we discuss RVAF architecture and in Section V its formalization. In Section VI we present our experimental results on a conveyor-belt scenario. Finally in Section VIII we draw our conclusions and discuss future work.

II. AF: Arrowhead Framework

AF is the result of a set of EU projects in which SOA principles have been applied to IoT and Industrial IoT applications, respectively. As the main result of the Arrowhead projects, the AF continued its development in an independent fashion, and it is now being used in multiple industrial installations. It is further developed in other projects, like for example, the Arrowhead Tools project.

AF has what is needed to design, implement and deploy an Arrowhead SOA-compliant system. The objective is to allow all users to work in a common and unified framework, thus enabling a high level of interoperability. The AF includes three core services and systems [2], where the Orchestrator plays a key role in defining which application services are connected with each other, and the QoSM guaranteeing adequate levels of QoS for each connection.

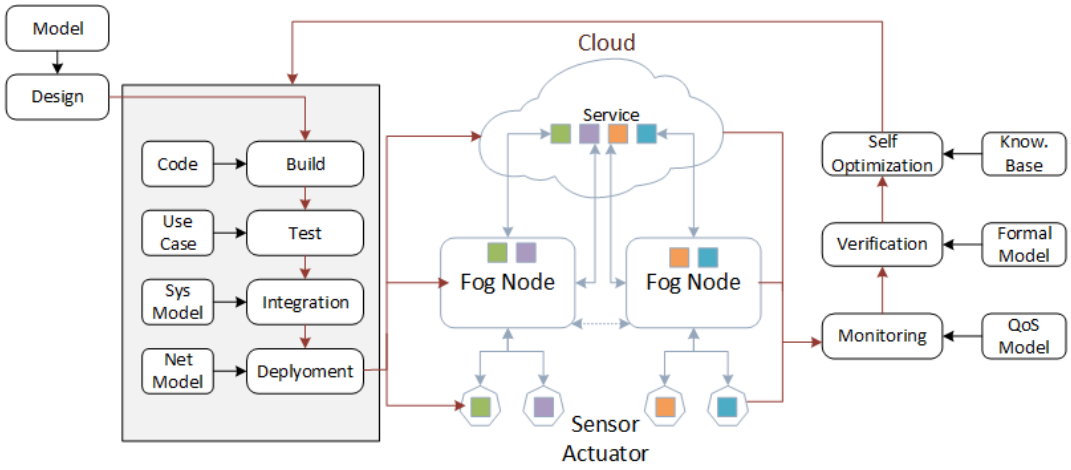


Fig. 1. RVAF-supported and IoT-based full-stack system architecture

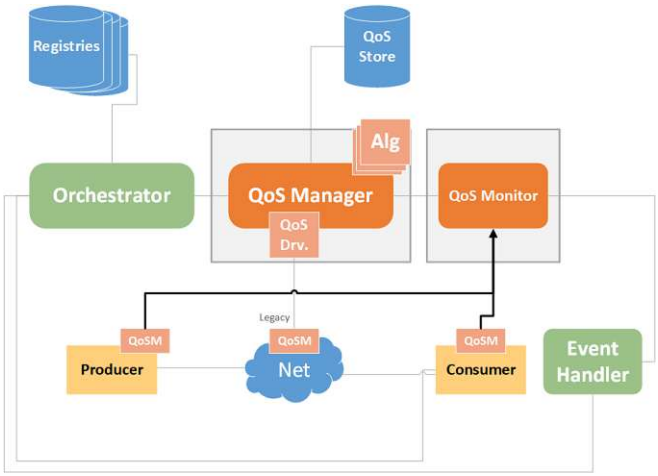


Fig. 2. QoS Manager Architecture

The AF builds upon the local-cloud (LC) concept, where local automation tasks are encapsulated and protected from outside interference. Application services usually communicate with services within the LC (intra-cloud orchestration), except when they communicate through the Gatekeeper Service with other AF-compliant LCs (inter-cloud orchestration). Each LC must contain, at least, the three mandatory core systems: Registry, Authorization, and Orchestration. They allow to establish connections between AF-application services. The core systems are accompanied by optional supporting systems, that further enhance the capabilities of an LC. The Event Handler is one of these systems, capable of implementing a producer/consumer communication pattern on an SOA logic.

III. Related Work

As shown in Figure 2, the QoS Manager (QoSM) in Arrowhead interacts with the Orchestrator and the QoS-Monitor, to ensure that connections between application-service instances, respect given QoS requirements. The QoSM handles information regarding the network topology, device capabilities, QoS requirements, etc, residing in the QoS Store (and other databases of the AF), accessible through an SOA interface. The management of the QoS is mainly concerned with the reservation of communication and computational resources. Their relevant information is maintained in a QoS Store. The QoSM is able to determine if the requested QoS can be granted or not, and configure the involved devices and active network elements (e.g., router and switches), to grant a QoS request.

The elements that can be configured by the QoSM include traffic-smoothing filters (on the output of service producers or consumers), traffic parameters (like traffic priority), and delivery guarantees of message-oriented middle-ware with QoS capabilities (like DDS [3], RabbitMQ [4] or XMPP [5]).

The Alg module contains different algorithms for QoS verification (examples will be given Section V, as Algorithms 1 and 2, respectively). The configuration of network actives and nodes is done by the QoS Drv module. This provides the drivers for interacting with custom or proprietary protocols. Some applications might need to know the current status of the system, in order to be able to adapt to changing conditions. This is achieved by monitoring the network status with the QoS Monitor, and informing the interested parties (e.g., application services in charge of ensuring the robustness of the system) through the Event Handler. A detailed description of the QoS support in AF can be found in [6].

IV. RVAF: System Architecture

An IoT architecture can be split in three distinct scopes of operation (see Figure 1): the cloud, the fog (edge), and the swarm (sensors/actuators) [7]. Moreover, sensors and actuators provide us with the ability to extract data from the physical environment and to actuate upon its state. In the fog specific tasks are performed such as data aggregation, filtering, or real-time control. In the cloud we have ”unlimited” amount of storage and the ability to perform large-scale computation

within conceivable time. Development of IoT systems that spread over multiple scopes of operation is a challenging task. It involves several levels of development such as system or infrastructure services; user application services; and specification, configuration and validation services. We describe an approach on increasing QoS of IoT systems and making them more dependable in general by using it's own existing infrastructure for continuous integration and deployment (CI/CD). RVAF system can monitor IoT services, detect faults and errors, ensure service stability, reduce downtime and increase overall availability.

A. CI/CD: Continuous Integration and Deployment

Automation of system development and operation is achieved through a continuous cycle of development, integration, and deployment (CI/CD). This cycle ensures an automated evolution from code to full applications, and an automated maintenance and optimization, where newly built code is automatically integrated with the rest of the system and deployed to a target.

If the system is malfunctioning, the target code will provide feedback to the developer, such that potential bugs and errors in the system are corrected. CI/CD is therefore an essential component of distributed and heterogeneous systems, such as a full-stack IoT-based systems. Being part of the infrastructure that supports the development and maintenance of IoT platforms and applications, CI/CD contributes to a reduction of development time and costs, and ensures a higher reliability and robustness of a system.

Arrowhead is an open-source project [8] and it was mainly using a centralized development method together with custom designed manual deployment methods implemented by the end user. In this paper we also introduce an automated continuous integration and deployment system for Arrowhead core services that can be implemented with the CI/CD system at the application level. It provides a novel approach to integrate user level services with core services in a fully automated way. This ensures that not only user level services, but also, system level services are kept updated. As shown in Figure 3, RVAF services are deployed through the use of containers, such as the ones provided by the Docker Swarm or Kubernetes. Each service is installed in a single container with all its required dependencies. Containers are pre-built and stored in a container-image registry, the Docker Registry, which is maintained automatically by the continuous integration system (CI). In the example shown in Figure 3, the CI is implemented using the Jenkins CI tool [9]. It contains a build server for Arrowhead core services and user level services. Each service needs authorization certificates to join an Arrowhead local cloud, each service needs to be configured and provide endpoint information to the rest of the system. This configuration process is performed using Infrastructure as Code and deployed by Jenkins pipelines. The actions of integration and delivery to the Docker Registry are automatically triggered by major version commits in Arrowhead code repository.

An RVAF continuous-deployment system (CD), connects the images of services with their respective targets (fog nodes, cloud virtual-machines), and deploys them accordingly. The CD is responsible for updating the containers in case a service changes in the Docker Registry. It allows to update services on demand, to rollback services if necessary, or to remotely control services or clusters of services. Container orchestration is observing container parameters such as CPU and memory usage, however this is not reflecting a functional health state of a service within the container. To ensure QoS on both levels it is necessary to observe certain parameters within the service that could affect its ability to deliver correct functionality.

B. RM: Runtime Monitoring

A further benefit of RVAF container-based deployment, is the ability to monitor at runtime (RM) each container, in terms of resources or application-specific parameters. RM is performed either on a target node, or remotely. As each container is identified with a single service, one can single out properties of a specific service, and monitor it individually. RM thus allows to observe the behavior of each service, and evaluate the health of the entire system.

In the context of RVAF-supported full-stack IoT applications, we have to consider large scale multi-dimensional data streams, with multiple sources. To fully monitor such distributed applications, all sub-agents need to be observed simultaneously. As an RM infrastructure we therefore chose to integrate the Prometheus monitoring tool [10] with the QoSM. This combination supports real-time, RM capabilities on either a local machine, or in the cloud. RVAF uses soft real-time monitors wrapped around PromQL [10] queries to observe metrics from individual containers. The data collected from a service or container is exported into time series database, whereof it can be aggregated, filtered or monitored raw.

These features perform run-time monitoring and run-time verification of the individual services, metrics within the services, containers and whole clusters. It provides an additional and novel approach to ensure QoS in industrial IoT systems. It increases fault-tolerance, and can even be used to ensure resilience to attacks on an IoT system.

C. SO: Self-Optimization

The IoT applications considered are heterogeneous hardware and software systems. This and the fact that they are implemented at multiple scopes of operation, makes them very difficult to develop, manage, and operate. Thus it is important to design these applications with basic principles of autonomic computing, such as, self-adaptation, self-organization, self-configuration, self-protection, self-healing, self-description, self-matchmaking, and self-energy supply [11].

We identify self-optimization (SO) as the most relevant property for the QoS of IoT frameworks. SO is the ability of a system to take corrective measures, in order to maintain the optimal usage of constrained resources [11]. The concept of QoS management in AF, as described in [12], detects violations and informs the other core services about the

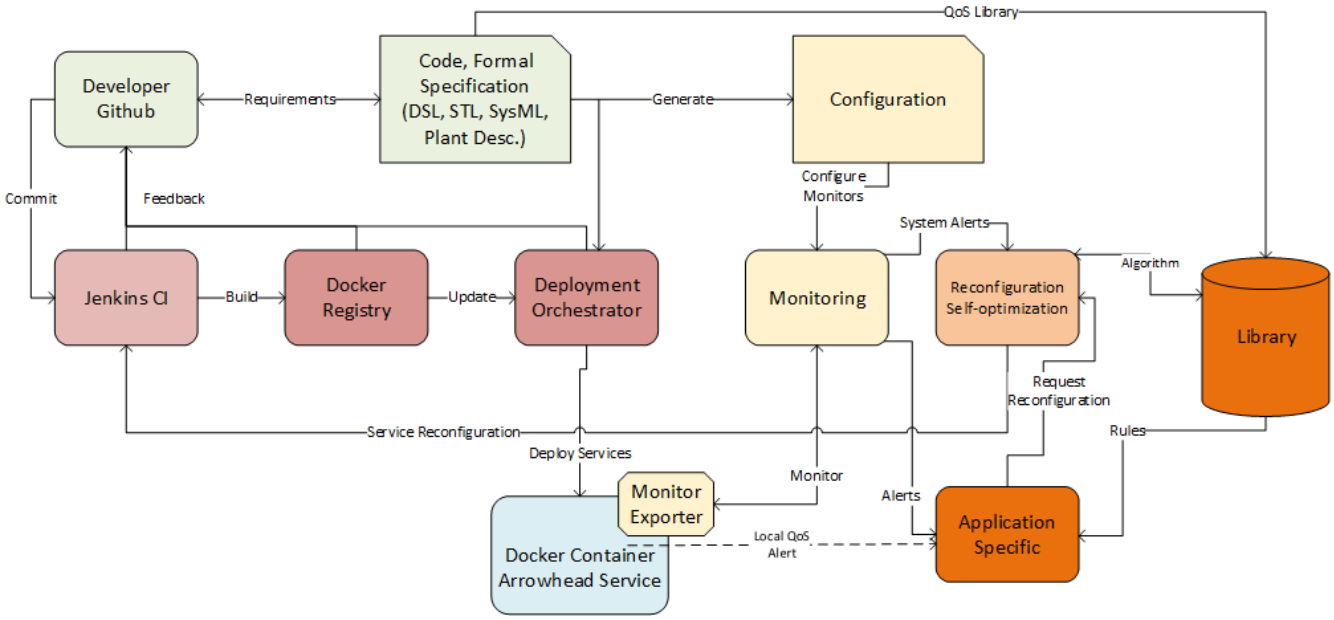


Fig. 3. RVAF container-based service deployment

violation, it is the responsibility of these core services to take any corrective, or containment actions.

In RVAF we extend the violation detection/notification ability, with the ability to take corrective actions, on services, Arrowhead clouds, or target nodes in general. This ability falls under the notion of SO, as defined in [11]. The RVAF-SO corrective infrastructure is realized by merging the AF-QoSM functionality, as described in [12], with the functionality provided by the RVAF-CI/CD and RVAF-RM described in the previous two subsections.

V. Formalizing the QoS Problem

A. System model

In its current form, RVAF is mostly suited for soft real-time IoT applications consisting of a set of interconnected services on the same local cloud and where multiple services can be deployed on the same node. Nevertheless, its principles can be easily extended to other scenarios and QoS properties.

As shown in Figure 4, an RVAF model considers the system to be composed of a set of N hosts (nodes) $H_1, ..., H_N$ and a set of M services $S_1, ..., S_M$. Services are interconnected through links $l_{x,y}$, characterising the connection between services S_x and S_y. The services S_x and S_y can be a typical Arrowhead SOA consumer/producer services, where each connection between a consumer or provider can have a set of QoS requirements. The QoS requirements are part of the formal specification of the code, and later on saved in the QoS Store.

Each node runs a Reconfiguration module R_x, which is mapped to the Jenkins CI tool, (see Figure 4), where x is the index of the node. Each module R_x can be connected to another Reconfiguration module R. The initial operation

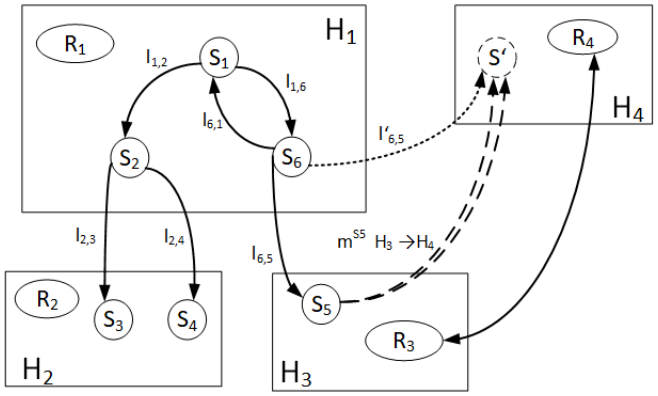


Fig. 4. System Model Example

occurs during service setup, where the Deployment Orchestrator module (in Figure 2) determines in which node the service can be deployed. This initial operation is denoted as $m^{S5}0 \rightarrow H_3$. As depicted in Figure 4, an operation $m^{S5}H_3 \rightarrow H_4$ represents the mobility/reconfiguration of service S_5, between nodes H_3 and H_4, respectively. In this case, H_3 represents the source node and H_4 represents the the destination node. Link $l'_{6,5}$ represents the connection that has to be established after the mobility operation is completed (rebinding). Consequently, connection $l_{6,5}$ will have to be safely deleted prior to $l'_{6,5}$ becomes operational. By safely, we mean that no messages should be lost or delivered to wrong nodes. This last requirement is ensured by the full implementation of SOA principles, where each transaction is independent from all other transactions.

B. QoS requirements formal definition

A formal definition of the QoS parameters for a service has to capture functional as well as performance requirements, such as the resources needed by the service to be deployed (e.g., sensors/actuators, CPU power, memory). Therefore, service requirements can be represented by the tuple:

$$SQoS_x = \{U_x, D_x, C_x, Crit_x, M_x\} \quad (1)$$

U_x is the required processor utilization when the service is running. Note that U_x can be a set of values which apply to different hardware platforms (nodes) where the service can be executed. D_x is the service deadline which has a different meaning if the service is a consumer or a producer. For a consumer, this deadline is measured from the moment in time when it is evoked, until it receives a response from the producer service. For a producer service, this deadline is the time that elapses from the moment the service if evoked by the reception of a message from a consumer, until the time it returns a response. $Crit_x$ represents the criticality level of the service, and M_x is the memory required by a service to run. Many other QoS parameters can be defined for each service, like the percentage of lost messages, communications bandwidth, etc.

None of the parameters defined in Eq. 1 are mandatory. The total absence of QoS parameters means that the service is best effort. Also note that the values for these parameters is obtained from the code specification. We also assume that it is possible to know the status of each of the system node, by consulting the QoS Store. The structure of the systems is available by consulting the plant description (which is not depicted in the figures). Each entry on the QoS Store in Figure 2 should contain at least the following data for each host H_x:

$$H_x : \{LS_x, LM_x\} \quad (2)$$

LS_x is a list of services with associated QoS requirements ($SQoS_x$). For each of these QoS requirements the QoSM will record the last R measurements on the LM_x list, which enables, as an example, to calculate the average, maximum, and minimum, and determine the tendency of the results. Each entry i of LM_x list is a measurement element Me_{x^i} related with service S_x. These values are used in Algorithm 2 to update the QoS requirements of the service on each node.

In this paper we focus only on (performance) QoS parameters like CPU utilization for the service, and memory consumption. We also assume that each service S_x requires a set of specialized resources in order to run, like access to specific sensors. Here we assume that service deployment is done with total independence from other services in a first come first served basis. In other words, if a service cannot be deployed to a node, then it will not be deployed into the system. In order to help the algorithm to decide where to deploy a service, a utility function is defined, which in this case is only based on two parameters, the percentage of CPU utilization (U_x) and the memory requirements (M_x):

$$U_x^H = f(U_x^h, M_x^h) = K1_x^h U_x^h + K2_x^h M_x^h \quad (3)$$

Eq. 3 uses parameters $K1_x^h$ and $K2_x^h$ to quantify the relative importance of CPU and memory utilization, respectively. The value U_x^H of the utility function is used to optimally assign a service to a host according to its requirements and resources needed. The pseudocode for service deployment is given in Algorithm 1.

Algorithm 1 RVAF Optimal service deployment

Input: List $\mathcal{H}$ of possible hosts H in the SoS, and a service S_x to be deployed

Output: The host H with maximum utility U_x^H where S_x should be deployed

```
1: L = NULL //list of hosts H matching the deployment requirements
2: for all H ∈ H do
3:    F = CI("VerifyServiceRequirements",H,S_x) //are the requirements satisfied?
4:    if F == true then
5:       Append(L,H) //add host H to the list L of hosts matching S_x requirements
6:    end if
7: end for
8: H = argmax_{H∈L} U_x^H //get matching host with best utility or NULL if L is empty
9: return H
```

If the service's requirements match the host's properties the service can be deployed to the selected host. Service deployment is controlled by a global container orchestrator or a custom reconfiguration algorithm that triggers a set of tasks in the CI/CD system for the deployment of the service S_x to a host H_x. In the example shown in Figure 3 the Service Deployment algorithm instructs the Jenkins CI tool to deploy service S_x to the Host H_x, and push the new configuration to the deployment orchestrator.

The container network on the H_x will be updated with the addition of service S_x. The capabilities of the AF allow services to be migrated from a host to another while maintaining full functionality. Moreover, local AF clouds can share services through secure connections. The deployment process on an infrastructural scale can be regarded as a technicality. However, service reorchestration at the application level needs to be coordinated with the local AF orchestrator service, and with the AF orchestrator service on the destination host H_x. The AF is designed to allow this kind of dynamic service orchestration [13].

C. Algorithm for run-time adaptation

The problem of deriving the correct QoS parameters, like processor utilization and memory footprint, is very difficult in open IoT systems, where the burden of finding out what is a predictable behaviour for a service, prior to run-time, can have a very high cost to the system developer, or it can even be an impossible task. This problem is mainly due to the following factors:

Algorithm 2 RVAF Run-time monitoring and adaptation: Migration Action

Input: A measurement M of the monitored parameters related to service S_x

Output: System-redesign alert or a migration $m^{S_x} H_a \rightarrow H_b$ of S_x from H_a to H_b

1: M = RM("getMonitoringData",H_a,S_x) //get monitoring data for S_x from RM
2: F = QoSM("UpdateQoSdata",H_a,M,S_x) //get violation flag from the QoSM
3: **if** F == True **then**
4: RM("QoSViolationAlert",H_a, S_x) //publish violation alert to RVAF services
5: $\mathcal{H}$ = CI("getHostList") //get the current list of hosts in the system
6: H_b = Algorithm-1($\mathcal{H}$, S_x) //get the optimal host for migration
7: **if** H_b != NULL **then**
8: CD("migrate",H_a,H_b,S_x) //tell CD to migrate S_x from H_a to H_b
9: **else**
10: RM("SystemRedesignAlert",H_a,S_x) //publish redesign alert to service S_x
11: **end if**
12: **end if**

i Different programming languages and/or algorithmic constructs can be employed whose behavior in time may be hard to predict;
ii A service can be deployed on different hardware platforms and this can be accomplished with different hardware configurations;
iii No prior knowledge is available on how other application/services interfere with the behaviour of the newly developed service.

The solution in RVAF is to continuously monitor the execution data related to a particular service, in order to derive its appropriate QoS parameters. This is of utmost importance for services deployed on hosts, where no data is available in advance, for the execution of the given service. A simple solution is described in Algorithm 2. The function Update QoS data on H_x can use a very simple algorithm, like calculating the weighted average of the last N values, or it can use more sophisticated algorithms, capable of detecting outliers and filtering out the data. These solutions deserves further investigation.

The results of Algorithm 2 can be used by the RVAF-SO to trigger the migration of service S_x from host H_a to host H_b. In case the service deployment fails, when executing Algorithm 1, then a system redesign requirement is signalled to all subscribed hosts. In this extreme case, the application itself fails, and the overall system must return to design phase, as described in Figure 1.

In addition to migration, RVAF is capable of using various other actions, such that the IoT application best adapts to various scenarios. For example: reduction or expansion of available resources, version rollback, and service termination. This requires an extension of the decision process of RVAF-SO, based on the list $\mathcal{A}$ of available actions A. For each action A and failure F, we assume that SO can compute a decision weight w_A^F according to the current system load:

$$\forall A \in \mathcal{A}, \forall F \in \mathcal{F}, \exists w_A^F \in \mathbb{R} \tag{4}$$

Algorithm 3 shows a simple action-selection algorithm for RVAF-SO. This algorithm is supposed to be called by Algorithm 2, right after the QoS violation is detected. Algorithm 3 will then choose the optimal action for the specific failure F of Service S_x, with "migration" as a default option. This SO ability in RVAF can be extended with other self-awareness mechanisms, such as GAMS [14].

Algorithm 3 RVAF Action evaluation: Picking optimal action

Input: QoS violation alert F for measurement M of service S_x

Output: Optimal action A for failure F

1: $\mathcal{A}$ = CI("getActionList",F) //get all available actions associated with failure F
2: **return** argmax$_{A \in \mathcal{A}}$ SO("getWeight",A,F) //return action A with best w_A^F

VI. Implementation Results

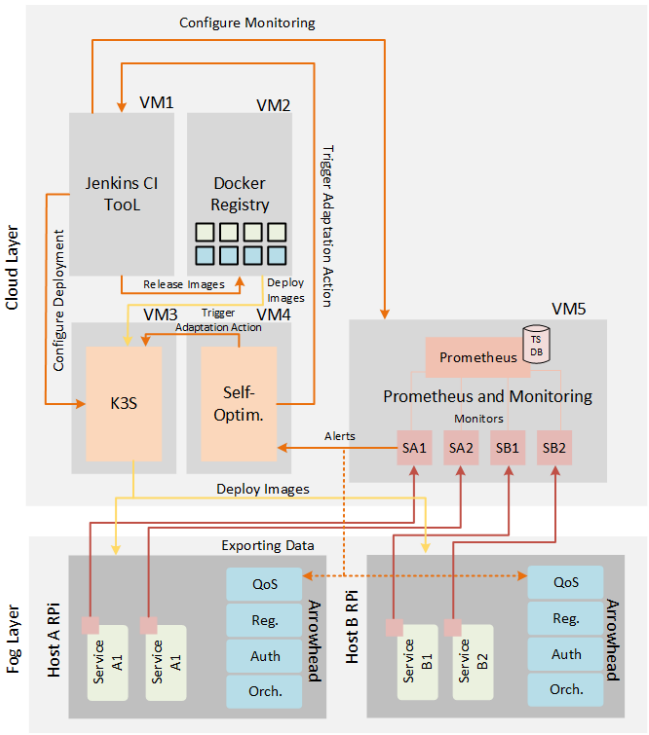


Fig. 5. Component diagram of the implementation

To illustrate the usefulness and versatility of RVAF, we implemented an industrial conveyor-belt application (CBA) with sorting capabilities. The CBA receives information from three induction sensors located on the belt. This information is aggregated in a fog node, in form of a sensor data aggregation service. This can be consumed by a sorting service, located either in the same fog node, in another fog node, or in the cloud. The sensors tell the sorting service, whether an object on the belt is metallic or not. If it is metallic it should be discarded from the belt. This is achieved by the sorting service by using a motor-driven actuator. The sensors and the actuators are interfaced to the CBA services through a soft PLC and an OPCUA server, both located on a fog node.

In Figure 5, we show a possible IoT-based realization of this application, with a redundant fog node, which is capable of replacing the main node, in case of failure, or QoS violation. Each fog node runs a local AF system, and a number of services, including the ones controlling the conveyor belt. The cloud consists of four virtual machines, which are allocated to the RVAF components as follows: CI in VM1, CD in VM1-VM3, RM in VM4, and SO in VM3. The Jenkins tool is used to implement the CI, the Docker Registry is used to store service-container images, the K3S is used as the deployment orchestrator, the SO is our reconfiguration platform given by Algorithms 1-3, and the Prometeus tool is used for data-logging and monitoring purposes. In order to monitor various services concurrently, we use a set of monitors, such as SA1-SA2 and SB1-SB2. The monitoring code is either automatically generated from the associated STL specifications, or written directly in PromQL. For the fog nodes we selected the Raspberry Pi platform.

Services are deployed in a single Docker container per service, and can be individually started, stopped, migrated or rolled back via a deployment orchestrator. Originally we used the Docker Swarm deployment orchestration, however due to the application in IoT, limited resources and development state of the tool we found K3S [15] to be much better suitable for the given task. K3S is a lightweight version of Kubernetes [16] designed for embedded devices and IoT. Each individual service can be observed on the application and container level, on top of the QoS management provided by K3S we add a an application level QoS management in combination with Arrowhead QoS manager.

Figure 6 shows a memory leak trace of a service in the system given in Figure 5, and its migration to the redundant host. It is in the nature of SOA approaches, that the development of services is done by different individuals and organizations. Thus, SOA approaches are error prone. However, the isolation capabilities of container-based deployment, allows us to detect faults and perform corrective actions with least amount of disruption for the entire system. The legend shows several metrics, where the last two (marked in bold) show the memory consumption in two containers. In the considered scenario, the memory leak in one of the services can be disruptive, and deny other services valuable resources, for reaching required reaction time. Mitigation by migration allows us to restart the service on the other fog node, where such behaviour can be better tolerated.

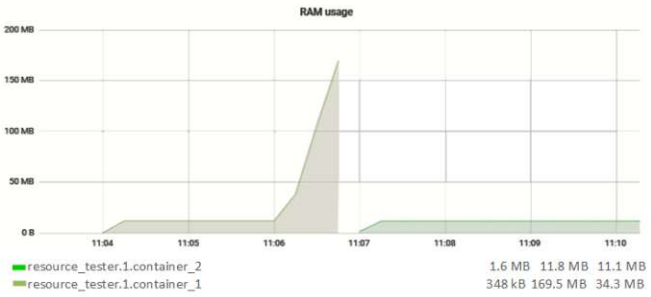


Fig. 6. Service migration results on memory leak

Stage	Average Time (s)
From detecting fault to orchestrator alert	0.82
Terminating faulty service	0.25
Terminating container	1.91
Redeployment of the container	1.74
Service activation with full JVM stack	3.49

TABLE I
TIME OVERHEAD ON SERVICE REDEPLOYMENT

Table I shows time measurements on different stages of the redeployment process. The response times can be further optimized by reducing overhead from different unnecessary functions in the used tools. As a example service we used a consumer template provided by Arrowhead. It is implemented in Java, hence longer startup time. This can be significantly reduced by using optimized service implementation. In case of migration of services from one node to the reaction times are similar to the ones showed in the I. An additional overhead can be created by using internal K3S scheduling mechanisms for migration. It will try to ”save” the node if possible before resorting to migration, this can create additional overhead on the process. This can be also reduced by configuring K3S for specific time limits during deployment or migration.

VII. FUTURE WORK

Various runtime monitoring tools have been developed over time, and they use domain-specific languages, to define individual monitors [17], [18]. These languages typically have a formal semantics, such as the signal temporal logic (STL) of [19]. They allow to verify during runtime, if the observed (monitored) behavior of the system, satisfies its given specification, standard, or regulation.

In RVAF we plan to introduce a translation system to formally capture performance properties (e.g., CPU and Memory usage) and functional properties in STL, query the multi-dimensional data acquired from a variety of sources with PromQL, and check their satisfaction. As an example, suppose that the CPU usage of the service Sx in an application A cannot exceed 80%. This can be formally specified in STL, and thereafter be compiled to a runtime monitor containing PromQL language queries, as follows:

STL:

$$A \models G\ (Sx[t] < 80)$$

PromQL:

```
sum(rate(cont_cpu-total_sec="Sx" [1m])) /
count(node_cpu_seconds_totalmode="A")*100<80
```

In future work we also plan to improve AF quality of service manager (QoSM), by fully integrating the the RVAF-CI/CD with the AF-QoSM. We also plan to harmonize the RVAF orchestration-service with the one of AF, within the Arrowhead local cloud or cluster. Finally, we plan to extend the library of actions that are made available for self optimization to the RVAF users.

In particular, AF local clouds have an orchestration service maintaining application level interfaces between services. This ensures the security and stability of the whole ensemble. Maintaining stability during RVAF adaptation, requires the harmonization of RVAF-SO with AF-QoSM and AF-Orchestrator, within the cloud. This can be performed at runtime in the following ways:

i Directly interface AF-QoSM with the RVAF-SO. The QoS violation alerts are sent to both simultaneously, and they agree on a mitigation strategy. This way the AF-QoSM is aware of any requirements violations, and is able to communicate potential changes to the Orchestrator. The Orchestrator updates local or remote cloud connections to the faulty service.

ii Rely on the self-awareness of the local AF cloud and perform mitigation actions for QoS violation, without informing AF-QoSM and AF-Orchestrator. This way, the consumer services are not aware of the changes, until the service is available and reintegrated in the AF system.

iii Rebuild a static configuration of the whole cloud/cluster, and restart the cloud with a new service configuration.

VIII. Conclusion

IoT applications are generally complex because they are often both distributed and heterogeneous. Moreover, the extension of IoT infrastructures to industrial domains, with high demands for dependability and security, requires the ability to monitoring and ensuring a certain level of quality of service.

This can be achieved by employing modern, continuous integration and continuous deployment tools (CI/CD), which are nowadays an essential part of the development lifecycle of any complex software artifact.

This paper introduced RVAF, an IoT-enabling and service-oriented (SOA) infrastructure, combining the QoS concepts of the Arrowhead Framework (AF), with the CI/CD deployment tools, runtime verification/adaptation, and the computational capabilities of cloud computing. RVAF supports the development of IoT systems that have the ability to: i) Monitor large scale applications in real time, ii) Verify and detect system properties, and iii) Perform self-optimization and service-redeployment during runtime.

Acknowledgements

This work was partially supported by the EU Horizon-2020 project ADEPTNESS under grant number 871319, the EU ECSEL-JU project Productive4.0 under grant number 737459, the AT BMBWF project CPS-IoT Ecosystem, the PT FCT/MEC project within CISTER Research Unit CEC/04234, and the PT ANI project Portugal-2020.

References

[1] "Plattform Industrie 4.0." [Online]. Available: https://www.plattform-i40.de/PI40/Navigation/DE/Plattform/Hintergrund/hintergrund.html

[2] P. Varga, F. Blomstedt, L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. Martnez de Soria, "Making System of Systems Interoperable The Core Components of the Arrowhead Framework," *J. Netw. Comput. Appl.*, vol. 81, no. C, pp. 85–95, Mar. 2017. [Online]. Available: https://doi.org/10.1016/j.jnca.2016.08.028

[3] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, May 2003, pp. 200–206, iSSN: null.

[4] Pivotal, "Rabbitmq." [Online]. Available: https://www.rabbitmq.com/

[5] XMPP, "XMPP." [Online]. Available: https://xmpp.org/

[6] L. L. Ferreira, M. Albano, and J. Delsing, "Qos-as-a-service in the local cloud," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[7] H. Isakovic, D. Ratasich, C. Hirsch, M. Platzer, B. Wally, T. Rausch, D. Nickovic, W. Krenn, S. Dustdar, and R. Grosu, "CPS/IoT Ecosystem: A platform for research and education," Oct. 2018, p. 8.

[8] M. T. Delgado, "Eclipse Arrowhead," May 2020. [Online]. Available: https://projects.eclipse.org/projects/iot.arrowhead

[9] Jenkins, "Jenkins." [Online]. Available: https://jenkins.io/index.html

[10] Cloud Native Computing Foundation (CNCF), "Prometheus." [Online]. Available: https://prometheus.io/docs/introduction/overview/

[11] S. K. Chaulya and G. M. Prasad, "Chapter 6 - Formation of Digital Mine Using the Internet of Things," in *Sensing and Monitoring Technologies for Mines and Hazardous Areas*, S. K. Chaulya and G. M. Prasad, Eds. Elsevier, Jan. 2016, pp. 279–350. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128031940000064

[12] M. Albano, P. Barbosa, J. Silva, R. Duarte, L. Ferreira, and J. Delsing, "Quality of service on the arrowhead framework," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, May 2017, pp. 1–8, iSSN: null.

[13] J. Delsing, *IoT Automation: Arrowhead Framework*. Boca Raton: Taylor & Francis Inc, Feb. 2017.

[14] S. Maksuti, M. Tauber, and J. Delsing, "Generic Autonomic Management as a Service in a SOA-based Framework for Industry 4.0," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, Oct. 2019, pp. 5480–5485, iSSN: 1553-572X.

[15] "K3s: Lightweight Kubernetes." [Online]. Available: https://k3s.io/

[16] "Kubernetes." [Online]. Available: https://kubernetes.io/

[17] S. Jaksic, E. Bartocci, R. Grosu, and D. Nickovic, "An algebraic framework for runtime verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2233–2243, November 2018.

[18] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*. Cham: Springer International Publishing, 2018, pp. 135–175. [Online]. Available: https://doi.org/10.1007/978-3-319-75632-5_5

[19] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Lecture Notes in Computer Science, 2004, pp. 152–166.

CHAPTER 8

An Energy Sustainable CPS/IoT Ecosystem

An Energy Sustainable CPS/IoT Ecosystem

Haris Isakovic[1], Edgar Azpiazu Crespo[2], and Radu Grosu[1]

[1] Technsiche Universität Wien, Vienna, Austria
[2] Mondragon University, Mondragon, Spain
name.surname@tuwien.ac.at
http://ti.tuwien.ac.at
edgar.azpiazu@alumni.mondragon.edu

Abstract. This paper provides a short overview on methods and technologies necessary to build smart and sustainable Internet-of-Things (IoT). It observes IoT systems in a close relation with data centered intelligence and its application in cyber-physical systems. With the current rate of growth IoT devices and supporting CPS infrastructure will reach extremely high numbers in less than a decade. This will create an enormous overhead on world's supply of electrical energy. In this paper, we propose a model extension for estimation of energy consumption by IoT devices in next decade. The paper gives a definition of CPS/IoT Ecosystem as a mutually codependent heterogeneous multidisciplinary structure. Further we explore a set of methods to reduce energy consumption and make CPS/IoT Ecosystem sustainable by design. As a case study we propose energy harvesting sensor node implemented as a wildfire early detection system.

Keywords: internet-of-things, energy consumption, sensor networks, energy harvesting

1 Introduction

The rapidly expending world of internet-of-things (IoT) is changing technological landscape for civil, industrial and social projects. It is projected that the number of IoT devices is going to reach 100 billion until year 2025 [40]. An average energy consumption of Raspberry Pi 3 devices is between 1Wh and 5 Wh [37]. To approximate average energy consumption of IoT application to about 4 Wh in the next 10 years we used following estimation formula, based on the models described in [7].

$$E_{IoT} = (E_{IoT_a} \times AR + E_{IoT_p} \times (1 - AR)) \times T \times D_{IoT} \times (\frac{100\% - ER\%}{100})^n \quad (1)$$

(E_{IoT_a}, E_{IoT_p}) are projected average consumption per device for an active and passive (i.e. sleep mode) operation.

(AR) provides the ratio between active and passive operation of a device, we chose three arbitrary cases as best, average and worst.
(D_{IoT}) is projected number of IoT devices.
(ER) is projected energy efficiency increase as proposed by [7].
(T) is time in operation over a year.
(n) provides projection period in years.

Case	AR	D_{IoT} (10^9)	$ER(\%)$	$T(h)$	Period	$E_{IoT_a}(Wh)$	$E_{IoT_p}(Wh)$	$E_{IoT_{2030}}(TWh)$
Best	0.01	100	5%	8640	10	4	2	**532**
Avg	0.3	100	4%	8640	10	4	2	**1493**
Worst	1	100	1%	8640	10	4	1	**3125**

Table 1. Different case scenarios depending for IoT device activity related to energy consumption.With the energy consumption projection of IoT devices in 2030.

IoT revolution will bring enormous influx of devices that are not always designed in energy efficient way as they are not considered as major energy consumers. However, if the quantity of these devices reaches the levels mentioned above it could create enormous overhead in energy consumption. Table 1 also provides three projections of energy consumption based on the Equation 1 in year 2030. In addition, there is energy consumed by cloud infrastructure as the number of cloud applications will increase accordingly. Cloud server infrastructures are already consuming enormous amounts of electrical energy, with up to 1.5% of total worldwide consumption [35]. With the influx of new IoT applications, data and emerging applications as a result it is evident that the cloud server capacity needs to be increased. Authors in [7] provide an estimate of cloud infrastructure energy consumption in range of 1000 TWh up to 8000 TWh. In addition there is a consumption required for communication. This approximation intends to show possible scale of the IoT energy consumption overhead. It is evident that energy efficiency of new devices will improve but the quantity of possible devices is still overwhelming. To put this in a perspective an average energy production of European Union (EU) is not around 3400 TWh over past couple of years [8]. Sustainability of the IoT applications and infrastructure depends on an energy efficient methodological design of IoT devices and applications, optimizing existing devices and applications, communal use of infrastructure, and energy harvesting capabilities.

In this paper, we are exploring energy aware design of IoT devices as a requirement for constitution of sustainable CPS/IoT Ecosystems. We define CPS/IoT Ecosystem as "a heterogeneous structure of hardware devices, and corresponding software components distributed over tree intertwined scopes of operation: cloud, fog/edge, and sensor/actuator nodes" [30]. These systems are distributed over different platforms and infrastructural facilities such as a power grid, the Internet, or a mobile grid. It is extremely difficult to maintain oversight on energy consumption over the whole composition. We propose technologies and

methods that can be used to ensure energy efficient and sustainable design from the perspective of sensor nodes. First, we will explore a set of methods that can be applied from the hardware design and up to ensure most beneficial performance to energy consumption ratio. Further, we will show examples on how to design a sensor node for IoT with low or completely neutral energy signature. Finally we will show how to optimize existing systems by introducing methods such as hardware acceleration.

In Section 1 we explore a problem of energy consumption overhead created by IoT devices and supporting CPS infrastructures. We propose an extended power consumption estimation model for IoT devices. Section 2 gives an overview of related activities on the topic. In Section 3 we provide a definition of CPS/IoT Ecosystem. Section 4 gives methodological steps towards energy aware CPS/IoT Ecosystem. Further, Section 5.5 proposes a sensor node that utilizes energy harvesting methods. Section 6 gives a short overview on a use case that can be realized with the described energy harvesting sensor node. Last two sections conclude the paper and provide due acknowledgements.

2 Related Work

The problem of the rapidly growing IoT market among with its energy needs, requires focussing on possible solutions for an efficient energy use on all the CPS/IoT Ecosystem's scopes. In this context, different approaches have been tried to reduce the energy consumption of sensor networks as, for example, trying alternative routing schemes to achieve a network lifetime increase as well [43]. Moreover, in traditional networks, the network configuration is not changed after initialization, but energy consumption improvements are noticed by adaptive network configuration where, if the distance between two sensors is calculated, the transmission power can also be adjusted avoiding the use of unnecessary power [48]. Further, analysing the architecture of a sensor, the main components that affect to their lifetime are identified so that aggressive energy optimization can be performed [38]. Nonetheless, the data centers used for cloud computing are an important point to consider regarding energy consumption. These centers can host up to several thousands of servers and they reach 1.1% to 1.5% of the total electricity use worldwide and is likely to rise [35]. In this context, an efficient cloud computing is crucial for which diverse hardware and software strategies can be adopted in all the levels that compose a data center [32][25][17]. An efficient operation can also be achieved for the Edge scope's components such as cloudlets balancing the workload across the nodes or adapting their configuration to manage latency and energy consumption [47].

In addition to managing the energy consumption, a wide range of energy harvesting methods can be used as an extra power source [28]. One of the main energy harvesting techniques is based on vibration energy scavenging. Its applications range from railway vehicles [26] to supplying hearing aids [6]. Other main method focuses on the solar energy which can be used to power sensors on both outdoor and indoor environments [22][39][27]. However, the energy source this

paper focuses on is the thermal by using thermoelectric generators. The usages cover industrial applications to recover waste heat from engines and increase the vehicle's overall efficiency [31][24][16] as well as using an aircraft's fuselage varying temperature to power autonomous sensors and perform diverse measurements [41]. As a human body constantly generates heat, several studies have also been carried out to take advantage of it to power autonomous wearable electronics like watches, medical devices or other types of sensors [34][33][12]. However, the need to make the energy scavenging unobtrusively limits the power that can be obtained.

These autonomous sensors can be placed in inaccessible places where battery replacements are not possible so they only rely on the temperature gradient across the thermoelectric generator to work. Additionally, a wireless transmission module can be added to their structure to track the measurements remotely [15][14][9]. These sensors will perform the programmed tasks as long as a minimum temperature gradient is achieved. Nevertheless, alternative storage elements like supercapacitors can be employed to manage the generated power and keep the system working even if this power is discontinuous [42]. A similar approach is presented in this paper, but an event detection circuit is introduced so the measurements are performed only under certain circumstances and the generated energy is just stored otherwise.

3 CPS/IoT Ecosystem

In Section 1 we structurally defined CPS/IoT Ecosystem, in this section we will give a motivational overview and functional description of this concept. The IoT allows us to connect physical environment with a digital infrastructure, collect its data and store it for the purpose of later or runtime analysis. CPS is a collection of practical and theoretical methodologies that allow us to interpret physical data, create models of physical environments using this data, recognize and extract emerging behaviours and use them to optimize system in question. We observe two concepts as separate disciplines but highly dependent on each other. We conceptualize CPS/IoT Ecosystem structurally in three scopes of operation:

Cloud The cloud infrastructure provides the ability to construct computational units based on computational performance or storage requirements in an automated and scaleable fashion. It can compute and store enormous amounts of data and made it available to large number of users at the same time. This is an essential requirements for the large scale data analysis. Cloud servers are located at remote locations and accessible exclusively through internet.

Fog/Edge The concept of fog and edge computing describes platforms with ability to perform computing and storage tasks in relative proximity to a physical environment and with extremely low response rates [11]. Fog hosts time sensitive tasks such as control loop execution where it is required to perform complex calculation based on historical data with low latency. Fog and edge devices are

located within the same facility and are communicating using local area network from one side and the Internet to communicate with cloud level if necessary.

Sensor and Actuator There are two basic types of devices that allow to interact with the physical environment, either by observing it or manipulating it. The sensors and actuators are implemented using low energy, low performance devices that are deployable in an imminent proximity to the observed object. The number of these devices is manifold of what is required on the upper levels and although they use less energy it needs to be considered based on the quantity. These devices are often placed in inaccessible locations with limited maintenance capabilities and their energy supply relies on energy storage devices such as batteries, or on their ability to harvest energy from the environment.

4 An Energy Aware CPS/IoT Ecosystem

CPS/IoT Ecosystem is a concept that allows to transform real data from a physical environment into valuable information, in order to increase efficiency of a system in different ways. This process requires number of layers of technology from hardware and software perspective. It relies on massive data collection and processing, learning statistical and mathematical paterns, applying optimizations and novel applications for forthcoming and legacy systems. The scale of this undertaking will require a massive number of devices from sensor to cloud level. In Table 1 possible energy consumption scenarios for IoT, plus overhead on Fog and Cloud infrastructure as shown in Section 1. This is why it is necessary to reduce its energy footprint by adopting various methods that increase overall efficiency and reduce energy consumption in all scopes of operation. In this section we will discuss methodological approach that would outperform commonly used approaches.

Energy aware design for CPS/IoT Ecosystem has multiple dimensions. It doesn't depend on hardware or software alone, it is rather a "full stack" problem starting from a system model to the final application artefacts.

Standardized IoT Model One of the major obstacles in development of IoT systems is lack of standardization. IoT is building on top of embedded programming that has a significant number of libraries. A model based approach to development of IoT would ensure use of verified code that can be configured between energy and performance optimal instance. A model based approach would be realized in multiple aspects such as hardware platform model, service model, network model, application model and eventually wrapper model to unify all above mentioned cases. This would increase oversight and traceability of the system, reduce maintenance cost and increase resource utilization.

HW/SW Co-design CPS/IoT Ecosystem applications are dependant on full range of devices from a cloud server with high-power CPU, to microcontrollers in embedded devices and custom designed hardware for networking solutions.

Use of FPGA or hybrid architectures is widely accepted, and its application in CPS/IoT applications is more than beneficial. Molding HW/SW platform to specific needs provides the advantages of a dedicated platform such as resource utilization and performance and flexibility of a COTS platform. Both IoT and CPS can profit from custom designed accelerators or IP blocks. Both ends of the CPS/IoT Ecosystem can benefit from a targeted application acceleration. Using hybrid architectures such as Intel Arria 10 [2] or Xilinx Znyq [3] applications have access to a standardized CPU architecture and FPGA device within the same platform. The FPGA is ideal for acceleration of mathematical tasks, network routing, security tasks or other application specific IP. Study presented in [29] showed massive acceleration for tasks such as matrix multiplication up to 5 times and reduced energy consumption up to 4 times. This comes with a cost of flexibility and programmability but with application of high level synthesis tools it can be significantly reduced.

Energy Harvesting IoT is a leading industry in propagation of energy harvesting solutions from a piezzo electric auto-charging switches to solar driven fog and/edge stations. By adding energy harvesting capabilities use of electrical power from grid can be significantly reduced or completely neutralized as shown in Section 5.5. Environmental sources for energy harvesting are [10]: a) mechanical energy in form of vibration or movement with examples of piezzoelectric generators, wind and water turbines, b) Solar energy generated by the sun with examples of photo-voltaic generators or thermal generators, c) Thermal energy generated by environment with example of thermoelectric generator (TEG), electromagnetic energy generated by surrounding electromagnetic fields and converted into electricity by electromagnetic induction.

5 Energy Harvesting Sensor Nodes

In this section we propose an energy harvesting sensor node for remote data collection over narrow-band communication network. It is a sensor device that can use any of the above mentioned energy harvesting sources and can be deployed in a remote place with little to none maintenance. The sensor device is designed to track a specific event such as abrupt temperature rise or extreme kinetic forces in the environment. It is an environmental friendly device with no chemical batteries and low energy storage.

The sensor device shown in Figure 1 is based on thermo-electric generator, which produces an electric power on a temperature gradient in the surrounding environment. It powers a microcontroller centered device over a power management unit (PMU). The power management unit uses a supercapacitor as an energy storage element and gives a stable regulated voltage necessary for MCU to perform its tasks. As an interface between MCU and PMU an event detection circuit is placed. It tracks outside event and releases power to the MCU when the stable voltage is reached. The MCU is further interfaced with a Sigfox transceiver and two sensors, a temperature sensor and a global positioning sensor. When the temperature of the environment reaches threshold it will generate

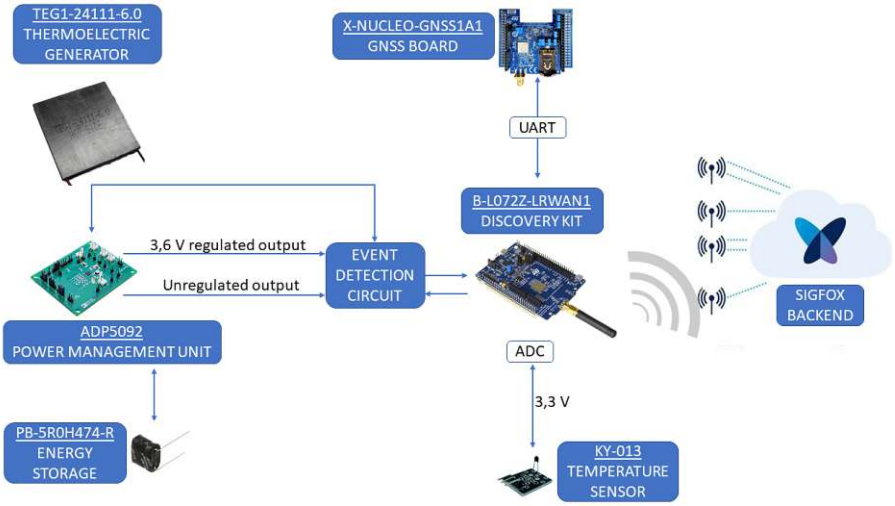


Fig. 1. Designed system's architecture

enough power to wake the MCU, acquire data from sensors and transmit the message over Sigfox network.

5.1 Thermoelectric generator (TEG)

The selected TEG device for this application is the TEG1-24111-6.0 manufactured by Tecteg MFR [36]. As high temperatures may be reached during an event detection and its processing, it is important that the TEG module can work in that range so the chosen module is designed with high temperature bonding materials allowing it to withstand temperatures of up to 320 °C and offers superior performance when its hot side is over 150 °C.

5.2 Power Management Unit

The output obtained from the TEG module is a varying irregular voltage so that it cannot be used to power the MCU. Therefore, a treatment stage is added making use of the ADP5092 [19]. This is an ultralow power energy harvester PMU which offers a wide range of configurations so it can be configured in accordance with the energy harvesting source and the connected load requirements.

The board offers a Maximum Power Point Tracking (MPPT) control which can set the maximum power point ratio for the energy harvester obtaining the maximum available power from it. This is configured in its dynamic sensing mode and adjusted for a TEG harvester.

Rechargeable batteries, capacitors or other storage elements can be charged with the obtained energy so it can later be used to power the load when needed.

In this case we are using the Eaton's PB-5R0H474-R [23] supercapacitor with 0.47 F and 5 V. Less environmental friendly option would be a battery supported device that can easily be integrated in the device.

The PMU provides two power outputs, an unregulated power output for charging of the supercapacitor, and a regulated power output for direct supply of the microcontroller. The unregulated output is connected to the storage element so its voltage changes with the supercapacitor's charge. The regulated output with a current limit of 150 mA is available, which is sufficient to run the MCU as described in Section 5.4. The output voltage is set to 3.6 V but can be regulated from 1.5 V to 3.6 V. The regulated output is used to power the load in this approach while the unregulated output is directed to the event detection circuit as explained in section 5.3.

Additionally, a couple of control features are used by the MCU to make a more efficient use of the PMU: RF interference are avoided by temporarily shutting down the boost regulator when a Sigfox message is going to be sent and the quality of the regulated output is checked before performing a cycle's actions.

In order to get an easy way to evaluate the ADP5092, the ADP5092-1-EVALZ evaluation board [21] has been chosen. This board provides a default working configuration which can be easily adjusted replacing any necessary component and making use of the provided jumpers.

5.3 Event detection

As one of the key steps is the event detection to know when to perform the measurements and data transmission, a circuit has been designed to ensure a correct detection.

In this case event detection is fully corelated with the energy generation event. This means that the sensor node is triggered when the enough energy is generated. To start energy generation TEG device requires a temperature gradient between exposed and isolated side of the device. The design of the device provide us with physical means to ensure the gradient necessary to generate enough power. For the current version a required gradient is 50 °C. This will correspond to aproximate 2 V regardless of the working temperature, so this is the threshold of the desired event. Figure 2 represents the designed circuit to detect events over the defined threshold.

The thermoelectric generator is connected to the PMU and a 3.6 V regulated output is obtained to power the board. The board should be powered just when an event is detected and shut down otherwise so to control this, a comparator based circuit has been designed. This comparator compares the TEG output to the defined event threshold. To get this fixed voltage, the unregulated output of the PMU is directed through a low-dropout (LDO) regulator. The regulated output is used as the positive power supply for the comparator while the negative supply is connected to ground.

When the TEG output is higher than the threshold, the comparator gives a low output. Otherwise, when the TEG output is below the threshold, a high

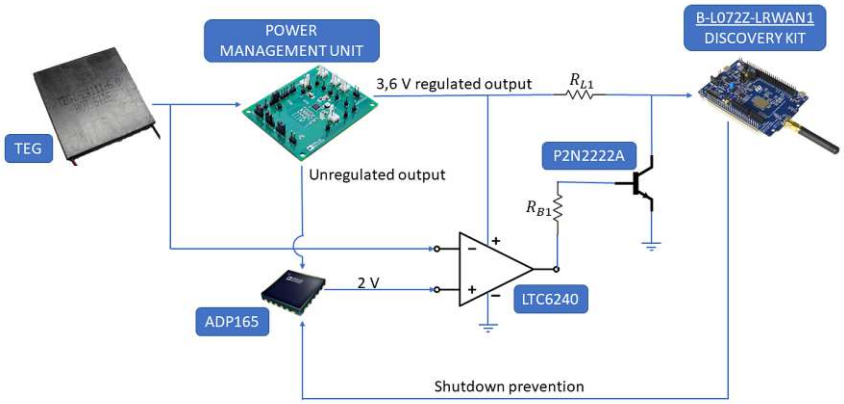


Fig. 2. Event detection circuit

output is generated. In order to control the board supply with this output, a transistor that acts as a switch is used. The comparator's output is connected to the transistor's base through a resistor (R_{B1}, so a low comparator output causes the transistor to work in its cut-off region leading to power the board, and a high output causes it to enter its saturation region resulting in a board shutdown.

Additionally, a shutdown prevention connection has been added to ensure the board is supplied until it finishes all its process. Otherwise, the TEG output could drop below the detection threshold causing a sudden board shutdown even if enough energy is still available in the PMU. This prevention is achieved by controlling the ADP165 voltage.

The ADP165 is a very low quiescent current, LDO, linear regulator [18]. Although an event detection threshold has been defined, the adjustable output option has been chosen over a fixed output to have some flexibility if any change is needed in the future. Figure 3 represents the used configuration.

To set the output voltage, R_1 and R_2 are adjusted. Additionally, the regulator can be enabled or disabled by means of the EN pin which is used for the board's shutdown prevention using a similar approach as before with a transistor. So, the regulator is working when the board is not powered but it is disabled after an event is detected. This means the regulator gives a low output forcing the TEG's comparator input to be always higher. The ADP165CP-EVALZ [20] evaluation board that simplifies the testing process has been used to try the configuration.

5.4 Microcontroller

The B-L072Z-LRWAN1 discovery kit [44] has been used in this project. It is a development tool which includes the CMWX1ZZABZ-091 open module by Murata allowing to use LoRa and Sigfox technologies. Being powered by an

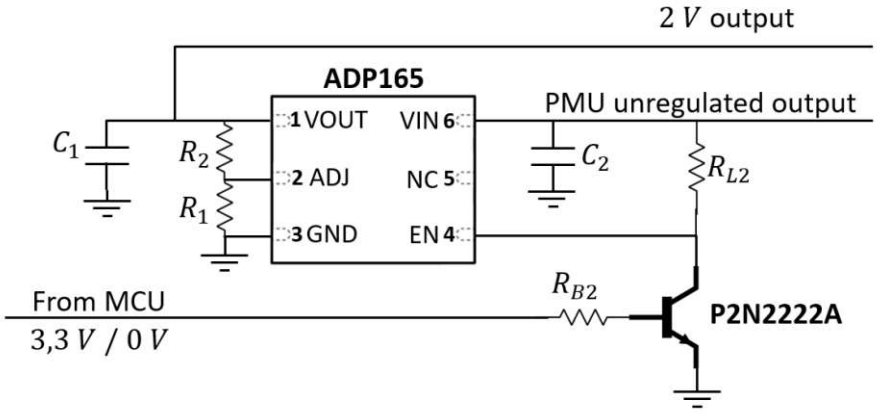


Fig. 3. ADP165 voltage regulator circuit

STM32L072CZ microcontroller which offers an ultra low power consumption makes it a great choice for IoT and energy harvesting applications.

Location acquisition One of the relevant data to be acquired is the current location of the board for which the X-NUCLEO-GNSS1A1 module has been used which represents an easy-to-use, GNSS solution [46]. Both UART and I2C interfaces are available to establish a connection but UART is used in this case. It is compatible with the Arduino UNO R3 connector also present in the B-L072Z-LRWAN1 making it easy to connect them and allowing to stack extra components.

Temperature sensor The temperature is another important parameter to track as it conditions the performance of the whole system. The sensor chosen to read it is the analog low power consumption STLM20W87F by STMicroelectronics [45]. However, the analog KY-013 module has been used for testing purposes. It consists of a NTC thermistor whose resistance changes with the temperature leading to an output voltage variation. The sensor is connected using the Arduino UNO R3 [1] connector of X-NUCLEO-GNSS1A1 since it is stacked. Then, the real temperature is obtained using the ADC.

Operating mode Different possible operating modes for the board have been identified. The chosen operating cycle is shown in Figure 4.

Increase in outside temperature creates a temperature gradient that generates electrical potential. When the power management unit detects this event it releases current and activates the board. This could be a short term event and take up to few seconds at most, so the energy is stored in the supercapacitor.

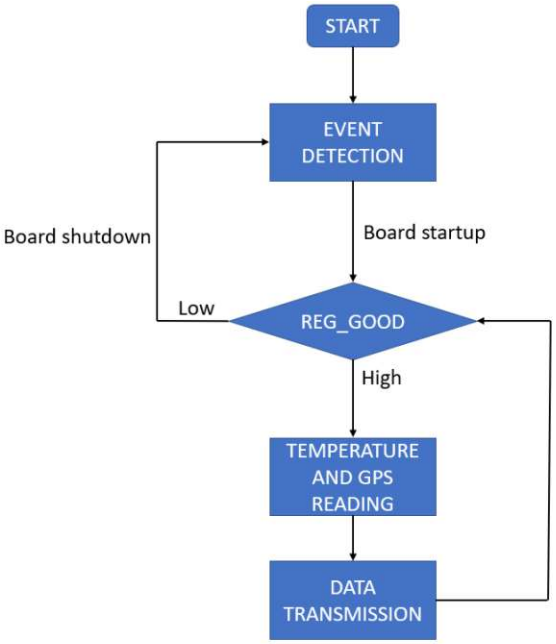


Fig. 4. Flowchart of the MCU's operating cycle

The power management unit releases the energy from supercapacitor to power up the board.

Once powered up, the quality of the supply voltage is checked making use of the REG_GOOD pin in the PMU. If a high signal is read, the board reads the temperature and location measurements to, finally, send the data. Otherwise, the board is powerd down and until next event.

As the detected event may still be active after the first data is transmitted, and some energy may be available yet, the measurement and data transmission steps are repeated until there is not enough energy to repeat the process or a low signal is read in REG_GOOD. Like that, a complete event tracking and analysis is done.

Another option that we considered is to enter the standby mode and stay there until an event is detected. At that point, an interruption that wakes up the device is triggered, and the measurements are performed before the device goes back to standby mode. The main reason for the cycle selection is the uncertainty of the available harvested energy amount. As it is not known when an event will happen, the amount of time the device can stay in standby mode is undefined and, although its power consumption is very low, it needs a continuous supply which may not be available. Taking this into account, the selected mode fits better, powering the board just to track an event once it is detected.

Power consumption A rather pessimistic estimation for the power consumption of each step as well as the time spent on it is shown in Table 2. It is worth noting that the estimation has been done considering a supply voltage of 3.3 V.

Power mode	Stage	Power consumption	Time	CPU/Bus frecuency
Run	Board start-up , REG_GOOD checking	8.12 mA	500 µs	32 MHz
Run	Location acquisition	33.11 mA	8 s	32 MHz
Run	Temperature reading	18.11 mA	25 µs	32 MHz
Run	Signal Tx	128.11 mA	2.1 s	32 MHz

Table 2. Execution cycle steps' power consumption.

Taking this data into account, an average current consumption of 52.86 mA is obtained.

5.5 Sensor node evaluation

Energy harvesting sensor node was implemented in an experimental setup. It was used to test individual behaviour of components and the sensor as a whole. However, the TEG output was simulated making use of a power supply.

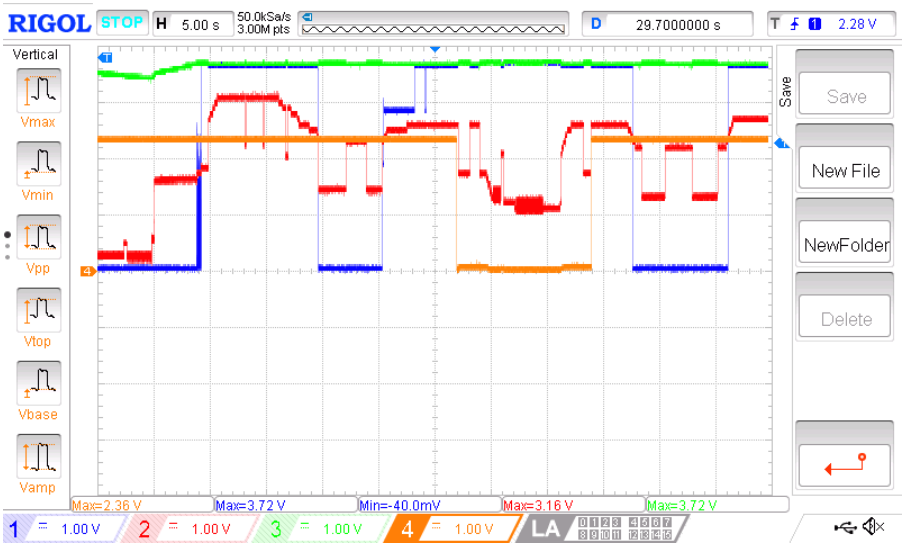


Fig. 5. System testing results

While testing the whole system, the following points were measured with an oscilloscope:

– Channel 1: Board power supply.
– Channel 2: TEG output simulation, V_{in}.
– Channel 3: Regulated output from the PMU.
– Channel 4: ADP165 output.

The captured results can be seen in Figure 5. First, V_{in} increases until it gets higher than the ADP165 output. The comparator makes the transistor work in its cut off mode forcing a high board supply. Then, V_{in} gets lower than the ADP165 output, so the board's supply gets low. Later, the ADP165 output drops to a low voltage when the shutdown prevention is activated. At this point, as V_{in} is higher, a high board supply is obtained always. After the shutdown prevention is deactivated, it returns to its initial operation.

If the shutdown prevention is activated, because the board is working, and V_{in} drops to low, the supercapacitor's voltage will also start to drop. When it discharges enough, the regulated output will go low forcing a low board supply as well. Whether the supercapacitor is charged, a high output will be present again. This behaviour is shown in Figure 6.

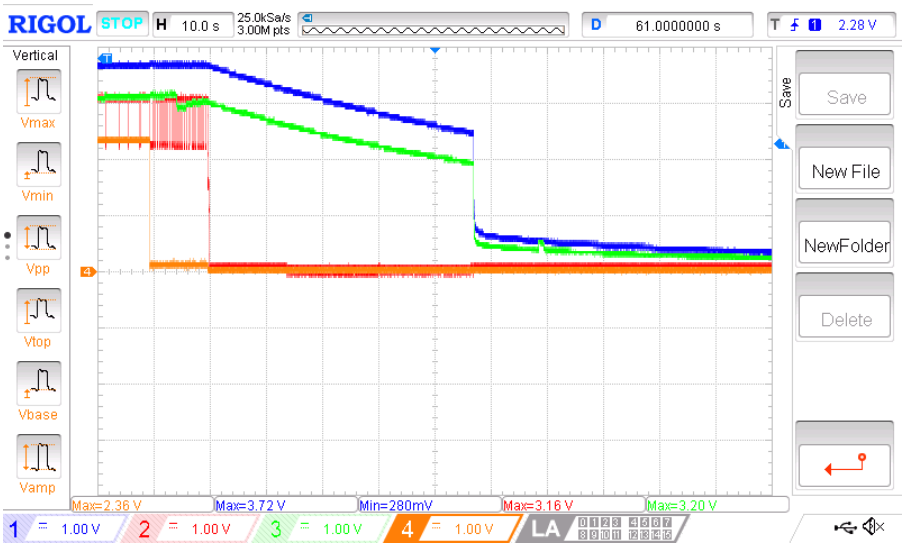


Fig. 6. System discharge testing

6 Energy Harvesting Use Case

The sensor node described in 5.5 can be applied in number of use cases. We are currently testing a prototype for a wildfire detection device. Despite all warning systems and services 100 people are confirmed dead during wildfires

in Greece in 2018 [4]. In United States in the state of California an area of 7664,39 km^2 was caught in one of the largest recorded wildfires in the US, 89 people are confirmed dead [5]. In addition, the economic and environmental impact of these natural catastrophes is colossal. The damages to infrastructure, public and private properties are measured in billions. Furthermore, damages to wildlife and environment are enormous and areas affected will require years to recover. The proposed wild-fire detection device would provide both localized and global early warning system for wild-fires. When combine with current early warning systems it would be able to increase precision and timeliness of detected fires on a large scale. With approximately 800 sensors we would be able to cover up to 50 km^2. In addition to satellite based detection systems such as [13] this could be extended even further. To cover the area burned down in California it would take about 160000 devices. However, it is not necessary to cover the whole area, instead it would be enough to create a mash based on geographical parameters or protect residential, industrial or agricultural zones. The cost for the deployment of wildfire energy harvesting sensor nodes is almost insignificant to the amount of damage created by the fire. The devices showed in Figure 7 can be placed on the endangered area with a distribution pattern that will complement other detection methods. This area should be covered with the low-band communication network, in this case we used Sigfox. The advantage of the radio based networks is the large coverage range and low power compared to other wireless communication networks. The device can stay dormant for years until a activation events occurs. In this case the activation event is a high temperature (or temperature gradient) generated by the fire. The sensor device will be triggered and sends a burst of alert messages containing temperature and location via narrow-band communication network to the cloud. Monitors in the cloud will notify authorities and provide them with the early detection necessary for a rapid response.

Figure 7 shows a package prototype of the energy harvesting node for the use case above. This design utilizes slow changing temperature of the ground in contrast to the air temperature, that changes rapidly in case of wildfire. The isolation layers divide hot and cold side of the TEG and ensure temperature gradient is achieved necessary to power up the sensor node. The cost of the prototype is given in Table 3 this can be reduced significantly in a serial production. This would make the platform highly viable and affordable considering the cost of the damages made by wildfires.

7 Conclusion

The number of IoT devices and supporting infrastructure used for CPSs is already affecting energy consumption and energy production models. This trend is continuing to grow and energy aware solutions are necessary to balance out the burden. In this paper, we proposed a model of estimating energy consumption for IoT devices and supporting infrastructure. We provided a set of methodological and design measures that could reduce energy consumption in CPS and IoT

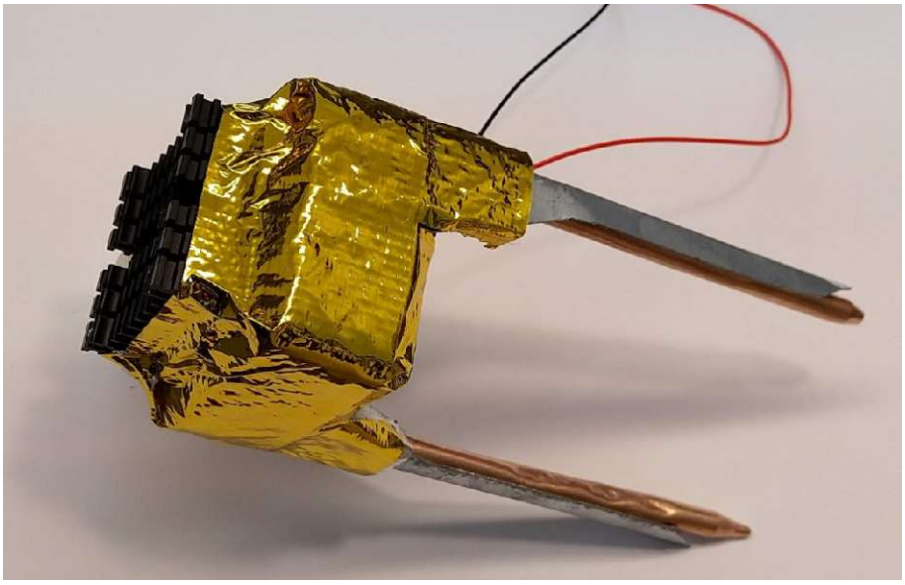


Fig. 7. Energy Harvesting IoT Node

Resource	Quantity	Unit cost (€)	Total (€)
TEG1-24111-6.0	1	50.00	50.00
X-NUCLEO-GNSS1A1	1	32.76	32.76
KY-013	1	1.00	1.00
B-L072Z-LRWAN1	1	43.52	43.52
ADP5092-1-EVALZ	1	44.00	44.00
ADP165CP-EVALZ	1	35.00	35.00
PB-5R0H474-R	1	4.47	4.47
LTC6240	1	1.89	1.89
P2N2222A	2	0.22	0.22
Various resistors	9	0.50	4.50
Case	1	25.00	25.00
Total			242.58

Table 3. Cost structure for the prototype.

significantly. Finally, we proposed an energy harvesting method that is simple to produce and could be used to increase effectiveness of active early wildfire detection systems. They could be used by individuals to protect their property and also by official institutions to protect certain agricultural or forest resources.

Acknowledgments

This work has been conducted within projects that has received funding from the Austrian Government through the Federal Ministry Of Education, Science And Research (BMWFW) in the funding program Hochschulraum-Strukturmittel 2016 (HRSM). This work is is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319.

References

1. Arduino Uno Rev3 — Arduino Official Store. https://store.arduino.cc/arduino-uno-rev3
2. Intel Arria 10 FPGA. https://www.intel.com/content/www/de/de/products/programmable/fpga/arria-10.html
3. Zynq-7000 SoC. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
4. Waldbrände in Attika 2018 (Jul 2019), `https://de.wikipedia.org/w/index.php?title=Waldbr%C3%A4nde_in_Attika_2018&oldid=190577400`, page Version ID: 190577400
5. 2018 California wildfires (May 2020), `https://tinyurl.com/y7nmnesb`, page Version ID: 956440002
6. Amor, N.B., Kanoun, O.: Investigation to the use of vibration energy for supply of hearing aids. In: 2007 IEEE Instrumentation Measurement Technology Conference IMTC 2007. pp. 1–6 (2007)
7. Andrae, A.S.G., Edler, T.: On Global Electricity Usage of Communication Technology: Trends to 2030. Challenges **6**(1), 117–157 (Jun 2015). https://doi.org/10.3390/challe6010117, `https://www.mdpi.com/2078-1547/6/1/117`, number: 1 Publisher: Multidisciplinary Digital Publishing Institute
8. BP: BP Statistical Review of World Energy 2017 p. 52 (June 2017)
9. Carmo, J.P., Goncalves, L.M., Correia, J.H.: Thermoelectric microconverter for energy harvesting systems. IEEE Transactions on Industrial Electronics **57**(3), 861–867 (2010)
10. Chalasani, S., Conrad, J.M.: A survey of energy harvesting sources for embedded systems. In: IEEE SoutheastCon 2008. pp. 442–447 (Apr 2008). https://doi.org/10.1109/SECON.2008.4494336, iSSN: 1558-058X
11. Chiang, M., Zhang, T.: Fog and IoT: An Overview of Research Opportunities. IEEE Internet of Things Journal **3**(6), 854–864 (Dec 2016). https://doi.org/10.1109/JIOT.2016.2584538, conference Name: IEEE Internet of Things Journal

12. Colomer-Farrarons, J., Miribel-Catala, P., Saiz-Vela, A., Puig-Vidal, M., Samitier, J.: Power-conditioning circuitry for a self-powered system based on micro pzt generators in a 0.13-μm low-voltage low-power technology. IEEE Transactions on Industrial Electronics **55**(9), 3249–3257 (2008)
13. Commission, E.: EFFIS - Active Fire Detection (Jan 2018), https://effis.jrc.ec.europa.eu/about-effis/technical-background/active-fire-detection/
14. Dalola, S., Ferrari, M., Ferrari, V., Guizzetti, M., Marioli, D., Taroni, A.: Characterization of thermoelectric modules for powering autonomous sensors. IEEE Transactions on Instrumentation and Measurement **58**(1), 99–107 (2009)
15. Dalola, S., Ferrari, V., Guizzetti, M., Marioli, D., Sardini, E., Serpelloni, M., Taroni, A.: Autonomous sensor system with rf link and thermoelectric generator for power harvesting. In: 2008 IEEE Instrumentation and Measurement Technology Conference. pp. 1376–1380 (2008)
16. David W. Kwok, James P. Huang, J.A.S.J.W.S.: US9018512B2 - Thermoelectric generation system - Google Patents (Apr 2015), https://patents.google.com/patent/US9018512B2/en
17. Dayarathna, M., Wen, Y., Fan, R.: Data Center Energy Consumption Modeling: A Survey. IEEE Communications Surveys Tutorials **18**(1), 732–794 (2016). https://doi.org/10.1109/COMST.2015.2481183, conference Name: IEEE Communications Surveys Tutorials
18. Devices, A.: ADP165 Datasheet and Product Info | Analog Devices, https://www.analog.com/en/products/adp165.html
19. Devices, A.: ADP5092 Datasheet and Product Info | Analog Devices, https://www.analog.com/en/products/adp5092.html?doc=ADP5091-5092.pdf#product-overview
20. Devices, A.: EVAL-ADP165-166 Evaluation Board | Analog Devices, https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-ADP165-166.html
21. Devices, A.: EVAL-ADP509X Evaluation Board | Analog Devices, https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-ADP509X.html#eb-overview
22. Dondi, D., Bertacchini, A., Brunelli, D., Larcher, L., Benini, L.: Modeling and optimization of a solar energy harvester system for self-powered wireless sensor networks. IEEE Transactions on Industrial Electronics **55**(7), 2759–2766 (2008)
23. Eaton: Eaton PB-5R0H474-R, https://www.mouser.at/datasheet/2/87/eaton-pb_supercapacitors-cylindrical-pack-data-she-1608804.pdf
24. Fernández-Yáñez, P., Gómez, A., García-Contreras, R., Armas, O.: Evaluating thermoelectric modules in diesel exhaust systems: potential under urban and extra-urban driving conditions. Journal of Cleaner Production **182**, 1070–1079 (May 2018). https://doi.org/10.1016/j.jclepro.2018.02.006, http://www.sciencedirect.com/science/article/pii/S095965261830310X
25. Gill, S.S., Buyya, R.: A Taxonomy and Future Directions for Sustainable Cloud Computing: 360 Degree View p. 68 (December 2018)
26. Güre, N.: Vibration energy harvesting from a railway vehicle using commercial piezoelectric transducers (2017), http://dspace.marmara.edu.tr/handle/11424/36590
27. Hande, A., Polk, T., Walker, W., Bhatia, D.: Indoor solar energy harvesting for sensor network router nodes. Microprocessors and Microsystems **31**(6), 420–432 (Sep 2007). https://doi.org/10.1016/j.micpro.2007.02.006, http://www.sciencedirect.com/science/article/pii/S0141933107000415

28. Harb, A.: Energy harvesting: State-of-the-art. Renewable Energy **36**(10), 2641 – 2654 (2011). https://doi.org/https://doi.org/10.1016/j.renene.2010.06.014, http://www.sciencedirect.com/science/article/pii/S0960148110002703, renewable Energy: Generation and Application
29. Hartberger, T.: Algorithm Implementation in HLS or HDL: Power Consumption and Efficiency Effects. Bachelor Thesis, TU Wien (Nov 2017)
30. Isakovic, H., Ratasich, D., Hirsch, C., Platzer, M., Wally, B., Rausch, T., Nickovic, D., Krenn, W., Dustdar, S., Grosu, R.: CPS/IoT Ecosystem: A platform for research and education. p. 8 (Oct 2018)
31. Jorge Martins, F.P. Brito, L.G.J.A.: Universidade do Minho: Thermoelectric exhaust energy recovery with temperature control through heat pipes (Apr 2011), http://hdl.handle.net/1822/15737
32. Kaur, T., Chana, I.: Energy Efficiency Techniques in Cloud Computing: A Survey and Taxonomy (Oct 2015), https://doi.org/10.1145/2742488
33. Leonov, V.: Thermoelectric Energy Harvesting of Human Body Heat for Wearable Sensors. IEEE Sensors Journal **13**(6), 2284–2291 (Jun 2013). https://doi.org/10.1109/JSEN.2013.2252526, conference Name: IEEE Sensors Journal
34. Leonov, V., Vullers, R.J.M.: Wearable electronics self-powered by using human body heat: The state of the art and the perspective. Journal of Renewable and Sustainable Energy **1**(6), 062701 (Nov 2009). https://doi.org/10.1063/1.3255465, http://aip.scitation.org/doi/10.1063/1.3255465
35. Mastelic, T., Brandic, I.: Recent Trends in Energy-Efficient Cloud Computing. IEEE Cloud Computing **2**(1), 40–47 (Jan 2015). https://doi.org/10.1109/MCC.2015.15, http://ieeexplore.ieee.org/document/7091782/
36. MFR, T.: TEG1-24111-6.0, https://thermoelectric-generator.com/product/teg1-24111-6-0/, library Catalog: thermoelectric-generator.com
37. Pi, R.: Raspberry Pi, https://www.raspberrypi.org, library Catalog: www.raspberrypi.org
38. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M.: Energy-aware wireless microsensor networks. IEEE Signal Processing Magazine **19**(2), 40–50 (Mar 2002). https://doi.org/10.1109/79.985679, conference Name: IEEE Signal Processing Magazine
39. Raghunathan, V., Kansal, A., Hsu, J., Friedman, J., Srivastava, M.: Design considerations for solar energy harvesting wireless embedded systems. In: IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005. pp. 457–462 (Apr 2005). https://doi.org/10.1109/IPSN.2005.1440973, iSSN: null
40. Rose, K., Eldridge, S., Chapin, L.: The Internet of Things: An Overview (Feb 2015), https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf, tex.howpublished: ISOC-IoT-Overview-20151221-en.pdf
41. Samson, D., Kluge, M., Becker, T., Schmid, U.: Wireless sensor node powered by aircraft specific thermoelectric energy harvesting. Sensors and Actuators A: Physical **172**(1), 240–244 (Dec 2011). https://doi.org/10.1016/j.sna.2010.12.020, http://www.sciencedirect.com/science/article/pii/S0924424710005182
42. Schlögl, P.: An Energy harvesting powered sensor node for machine condition monitoring. Ph.D. thesis (2018), http://repositum.tuwien.ac.at/obvutwhs/content/titleinfo/2962783

43. Shah, R.C., Rabaey, J.M.: Energy aware routing for low energy ad hoc sensor networks. In: 2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609). vol. 1, pp. 350–355 vol.1 (2002)
44. STMicroelectronics: B-L072Z-LRWAN1, https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html, library Catalog: www.st.com
45. STMicroelectronics: STLM20, https://www.st.com/en/mems-and-sensors/stlm20.html, library Catalog: www.st.com
46. STMicroelectronics: X-NUCLEO-GNSS1A1, https://www.st.com/en/ecosystems/x-nucleo-gnss1a1.html, library Catalog: www.st.com
47. Thomas Rausch, Philipp Raith, P.P.S.D.: Demo: A System for Operating Energy-Aware Cloudlets (Nov 2019), http://cpsiot.at/?p=235, library Catalog: cpsiot.at Section: News
48. Yan, R., Sun, H., Qian, Y.: Energy-Aware Sensor Node Design With Its Application in Wireless Sensor Networks. IEEE Transactions on Instrumentation and Measurement **62**(5), 1183–1191 (May 2013). https://doi.org/10.1109/TIM.2013.2245181, conference Name: IEEE Transactions on Instrumentation and Measurement

List of Figures

List of Tables

List of Algorithms