Deliverable D1.2

Application Design and Development Report

Editor:	Jag Minhas – Sensing Feeling
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public
Contractual delivery date:	September 30, 2021
Actual delivery date:	September 30, 2021
Suggested readers:	Researchers, developers and technologists interested in edge computing and the convergence of networking and computing.
Version:	1
Total number of pages:	39
Keywords:	Architecture, Distributed Computing, Protocols

Abstract

This document outlines the research and development work related to the Piccolo use cases outlined in Deliverable D1.1. These use cases have been refined on a technical and economic scale.

This work was done within the EU CELTIC-NEXT project PICCOLO (Contract No. C2019/2-2). The project is supported in part by the German Federal Ministry of Economic Affairs and Energy (BMWi) and managed by the project agency of the German Aerospace Center (DLR) (under Contract No. 01MT20005A). The project also receives funding awarded by UK Research and Innovation through the Industrial Strategy Challenge Fund. The project is also funded by each Partner.

Impressum

[Full project title] Piccolo: In-Network Compute for 5G Services
[Short project title] PICCOLO
[Number and title of work-package] WP1. Applications, Use Cases, Proof-of-Concept Prototypes
[Number and title of task] Tasks 1.2 and 1.3
[Document title] D1.2 Application Design and Development Report
[Editor: Name, company] Jag Minhas, Sensing Feeling
[Work-package leader: Name, company] Jag Minhas, Sensing Feeling

Executive Summary

This document outlines the research and development work related to the use cases outlined in Deliverable D1.1.

These use cases have been refined on a technical and economic scale, and the latest progress has been reported in the following chapters regarding:

- **In-vehicle Real-time Behavioural Risk Monitoring** – led by Sensing Feeling & Robert Bosch GmbH
- **Smart Factory** – led by Peer Stritzinger GmbH
- **In-Network Hardware Acceleration for Time-Sensitive Networking** – led by InnoRoute GmbH

Each of these use cases has sections discussing:

- Context & Goals
- Architecture & Design
- Development Status
- Challenges
- Next Steps

These reference points give a clear direction that the project is taking, explaining the work carried out thus far and giving defined steps of where changes need to be researched and developed.

List of Authors

Company	**Author**
ARM Ltd.	Chris Adeniyi-Jones
British Telecommunications plc	Adam Broadbent, Philip Eardley, Andy Reid, Peter Willis
Fluentic Networks Ltd.	Ioannis Psaras, Alex Tsakrilis
InnoRoute GmbH	Andreas Foglar, Marian Ulbricht, Surik Krdoyan
Robert Bosch GmbH	Dennis Grewe, Naresh Nayak, Uthra Ambalavanan
Sensing Feeling	Dan Browning, Jag Minhas, Chris Stevens
Stritzinger GmbH	Mirjam Friesen, Sascha Kattelmann, Peer Stritzinger, Stefan Timm
Technical University Munich	Jörg Ott, Raphael Hetzel, Nitinder Mohan
University of Applied Science Emden/Leer	Dirk Kutscher, Laura al Wardani, T M Rayhan Gias

Table of Contents

List of Figures

Abbreviations

API	Application Programming Interface
CAN	Controller Area Network
CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
I4.0	Industry 4.0
I/O	Input Output
IoT	Internet of Things
IT	Information Technology
MQTT	Message Queuing Telemetry Transport
NVR	Network Video Recorder
OBD-II	On-Board Diagnostics
OPC	Open Platform Communications
OPCUA	OPC Unified Architecture
OT	Operational Technology
PLC	Programmable Logic Controller
PoC	Proof of Concept
PTP	Precision Time Protocol
QoS	Quality of Service
RFID	Radio-Frequency Identification
RTEMS	Real-Time Executive for Multiprocessor Systems
RTSP	Real Time Streaming Protocol
SDK	Software Development Kit
TINC	Trusted In-Network Computing
TSN	Time-Sensitive Networking
VPE	Visual Processing Engine
VSS	Vehicle Service Specification

Definitions

Trusted Computing Base (TCB): An entire combination set of protection mechanisms within a computer system, including hardware, firmware, and software, responsible for enforcing a security policy.

TrustNode: Hardware accelerated Routing devices that include Field Programmable Gate Array (FPGA) for fast routing and supporting Central Processing Unit (CPU) for advanced packet processing. More information, see: http://TrustNo.de

RaspberryPI: Well known single board computer for rapid prototyping and research. Designed in UK. For more information, see: https://www.raspberrypi.org/

RealTimePI: Raspberry Pi with additional Real-Time HAT to enable hardware accelerated Time-Sensitive Networking (TSN) and time synchronisation features on the RaspberryPI. More information, see: https://innoroute.com/realtimehat/

1 Introduction

The aim of the Piccolo project is to advance technology in such a way that enables a smarter way to distribute computation, addressing growing concerns around factors including (but not limited to) privacy, latency and sheer quantity of data. Solving these issues gives more control to the end user about how their application is run, as well as removing the control that big cloud operators have over them. Alleviating these concerns therefore opens up new horizons for application development and supports new infrastructure markets. In previous documents (cf. Piccolo Deliverable D1.1 [1]), various potential use cases have been considered. Chapter 2 briefly explains which potential use cases were selected for becoming our Proof of Concept demonstrators, which are:

- **In-vehicle Real-time Behavioural Risk Monitoring**
- **Smart Factory**
- **In Network Hardware Acceleration for time-sensitive networking**

Chapter 3 goes into more detail of current progress of these Proof of Concept (PoC) prototypes, outlining the initial goals of the development, architectural considerations, along with identified challenges and next steps. We will also cover exploitability of the products, reflecting the commercial aspect that 5G has to offer, as well as the technical benefit of this and the new opportunities the technology has made possible.

Finally, Chapter 4 serves as a conclusion, summarising the key discoveries from the use cases thus far.

2 Deriving Application Proof of Concepts from the Identified Use Cases

In Deliverable D1.1 [1], many use cases were identified as solutions that could benefit from the technology discussed in this project. Some of these have had further research and development allocated to them, to draw them closer to a PoC state. A more detailed breakdown of each of these PoCs is described in the following chapter.

2.1 In-vehicle Real-time Behavioural Risk Monitoring

The concept behind this use case remains largely the same as in the previous deliverable. It combinines data from two sources, namely in-vehicle telemetry and vision sensing telemetry, to give an index of calculated risk in real time for the vehicle. With the abundance of sensors now available in vehicle, factors like acceleration, turning and velocity can be paired with visual based metrics such as nearby pedestrians or driver fatigue to give full flexibility over algorithm design. Having a quantifiable risk index can help identify potential problems for drivers which may be unnoticed otherwise, whether that is an alarm to the driver themselves or enabling more advanced fleet management. The factors considered for this calculation are mentioned in further detail in Chapter 3.

2.2 Smart Factory

Previously referred to as Plug&Produce for Industrial Production Systems, this PoC focuses on transporting workpieces between processing stations to ensure efficiency in fundamental parts of the production process. A novel core component in the scope of Piccolo is the dynamic distribution of software components (based on IEC61499) throughout the mesh network which controls the overall assembly line. This leads to far lower re-configuration times when the production setup is changed. The software manages its distribution and execution in a decentralized way, and can be injected into a system from any node. Further an intelligent workpiece traffic management is built on top of this distribution layer. A thorough overview is provided in Chapter 3.

2.3 In Network Hardware Acceleration for Time-Sensitive Networking

The IEEE 802.1Q standard defines technologies for deterministic communication on standard Ethernet. The technologies are summarized under the title Time-Sensitive Networking (TSN). By using Ethernet as the basic communication technology, all use cases can benefit from the resources available in the network for enhanced Quality of Service (QoS). These are for example jitterless and low latency packet delivery for industrial applications and low latency forwarding of Audio Video Broadcast (AVB) packets.

However, many people expect TSN to be plug-and-play like today's Ethernet, which works even without any configuration. But, this technology needs a complex configuration and dedicated hardware accelerators. While future Ethernet hardware will have the required accelerators built in, the control plane remains. As a step towards an easy-to-use TSN technology InnoRoute takes the special approach to consider a TSN switch as in-network Piccolo node configured via Piccolo Agent with the TSN hardware considered as in-network computing support.

This approach is investigated in the proof-of-concept, whereby the use case itself is not important, but the verification of the novel in-network node concept.

3 Proof of Concept (PoC) Development

3.1 In-vehicle Real-time Behavioural Risk Monitoring

3.1.1 Context & Goals

Behavioural risk monitoring in an automotive context involves identification of different factors (e.g. the route, traffic and weather conditions, driving behaviour etc.) that may result in critical situations like mishaps along with the probability of their occurrence. By predicting high risk events, mitigation steps can be undertaken like notifying the driver, changing the route or taking a short break to avoid increasing fatigue. Overall these measures can improve traffic safety by making concrete recommendations to the drivers and the passengers during a trip.

Computing risk indexes requires combined processing of a multitude of data streams (e.g., using different cameras such as interior or dashboard deployments, in-vehicle sensors such as acceleration sensors, etc.) along with publicly available information like weather and traffic conditions, and can, thus be compute intensive. In this PoC, we seek to exploit the computing resources from the vehicle to the infrastructure (using the Piccolo ecosystem) to evaluate the risk index. This brings many considerations including vehicular data access, trusted execution of behavioral risk models, privacy issues, etc.

3.1.2 Architecture & Design

The first architectural design of the In-vehicle Real-time Behavioural Risk Monitoring proof-of-concept prototype consists of different sets of components. Figure 1 illustrates the overall planned deployment architecture of the prototype. The risk indexes are computed based on the combined processing of multitude of data streams. For this, the prototype differentiates between two processing pipelines: (i) *vision data processing pipeline*, and (ii) *vehicular data processing pipeline*, both starting within the vehicle and transferred to multiple compute/processing nodes upstream (in the local Piccolo network and the cloud backend).

The following paragraphs provide more details about the two processing pipelines and the development status and the planned next steps.

3.1.2.1 Vision Data Processing Pipeline To begin designing the vision data processing pipeline, the first decision was how to create and setup the edge node that would be transmitting the data. In the current setup, multiple cameras are streaming their live feeds into a single Network Video Recorder (NVR). Using an NVR means that we can use wireless technology, avoiding potential added risk of cables navigating their way around the vehicle. This image acquistion system is based on Ubiquiti UniFi technology, boasting low powered cameras and a Cloud Key Gen2 Plus Controller

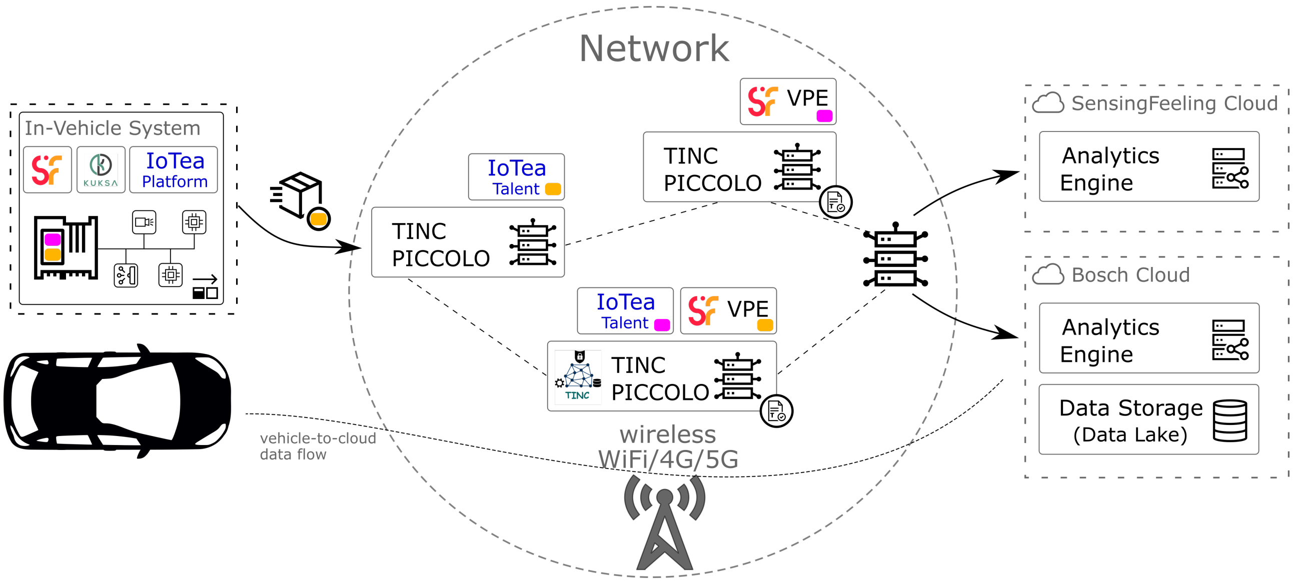


Figure 1: Deployment architecture of the Behavioral Risk Monitoring Proof-of-Concept.

which provides access to the virtual RTSP streams from the cameras.

The Sensing Feeling Edge node, which is also located inside the vehicle, can then connect up to the NVR's virtual streams and pull frames to convey to the Piccolo in-network Visual Processing Engine (VPE), in the Piccolo local network. Telemetry is then generated from each of these frames, and this raw data is then forwarded onto the Sensing Feeling Analytics Engine, hosted in the cloud, to carry out more computationally and state heavy processing.

3.1.2.2 Vehicular Data Processing Pipeline In-vehicular data (vehicle model, driving velocity, acceleration, steering movements etc.) serve as one of the essential inputs for any model evaluating behavioural risk as described in this use case. Setting up a vehicular data processing pipeline requires solutions to three questions:

- How to address/name the vehicular data attributes of interest? This is, in particular, complex on account of the high variance in the in-vehicular data, not only across automotive manufacturers but also within similar vehicle models.
- Which transport protocol to use for querying/fetching the data attributes from a vehicle?
- How to implement the business logic (pre)-processing the data attributes before feeding it to the risk evaluation model?

The choice of software stacks we use to set-up the data processing pipeline determines the answers to these questions to a great extent. We use *Eclipse Kuksa* [2], an open source software stack serving as a vehicular data access framework tackling the first two of the three aforementioned challenges, while the *Genivi IoT Event Analytics* (IoTea) [3] provides solution to the final question.

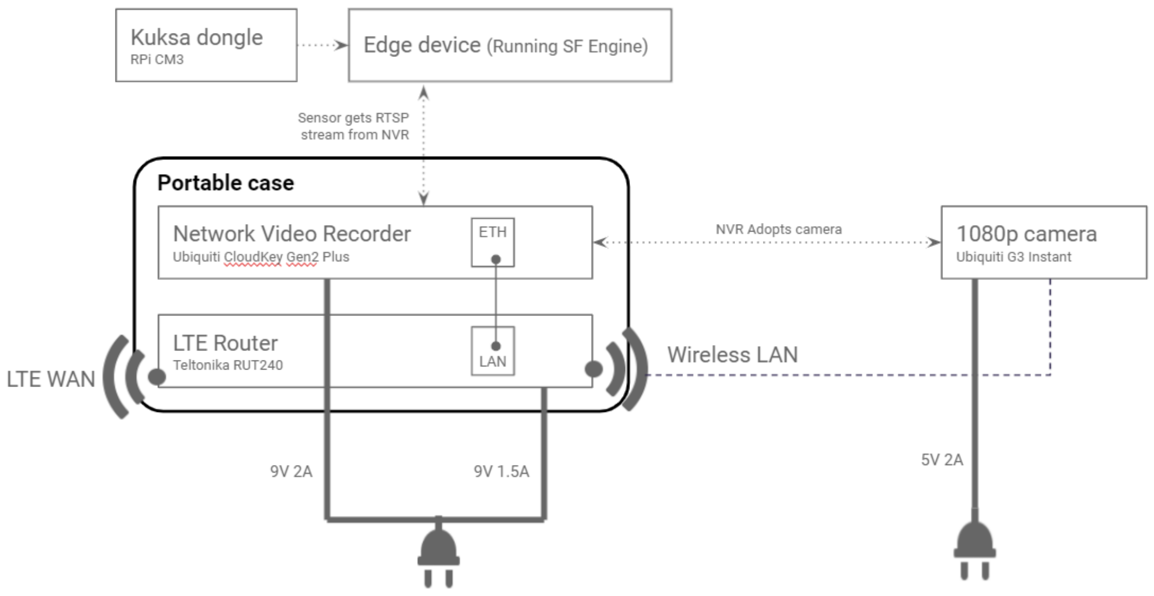


Figure 2: The current Edge Network Video Recorder kit, shown with a single camera

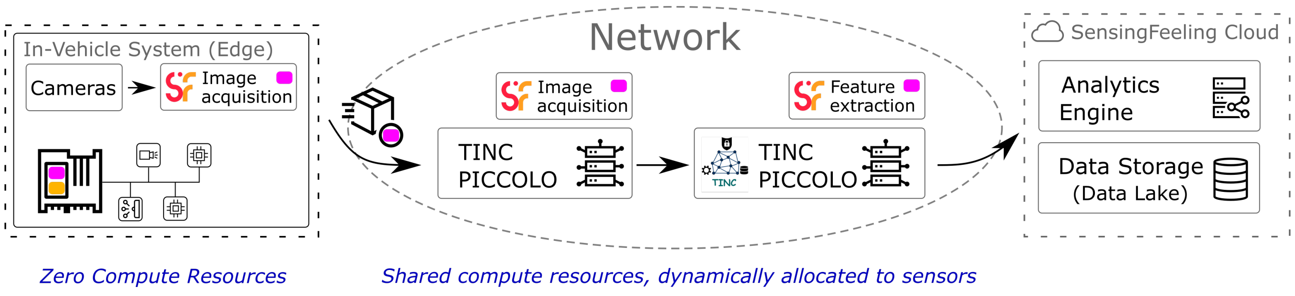


Figure 3: Piccolo's Vision Data Processing Pipeline. Data from built-in cameras are aggregated within the vehicle, while the vision processing will happen in the network towards the cloud backend.

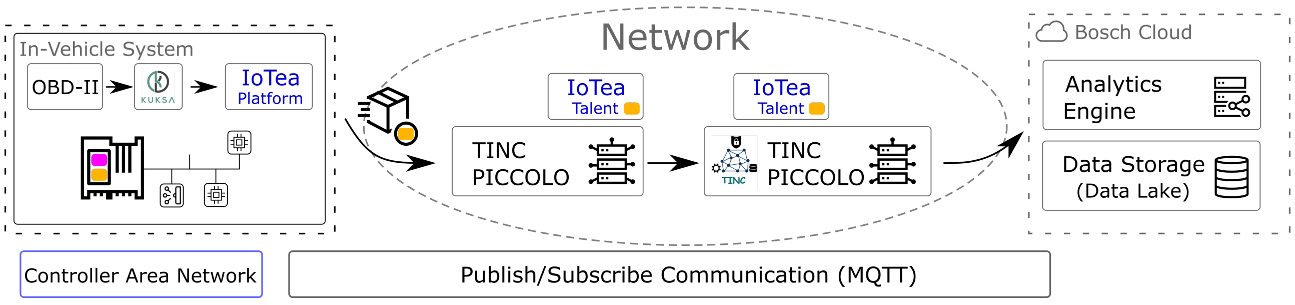


Figure 4: Piccolo's Vehicular Data Processing Pipeline. Data from the Controller Area Network is accessed via the OBD-II interface and made public available via Eclipse Kuksa and processed via IoTea Talents towards the cloud backend.

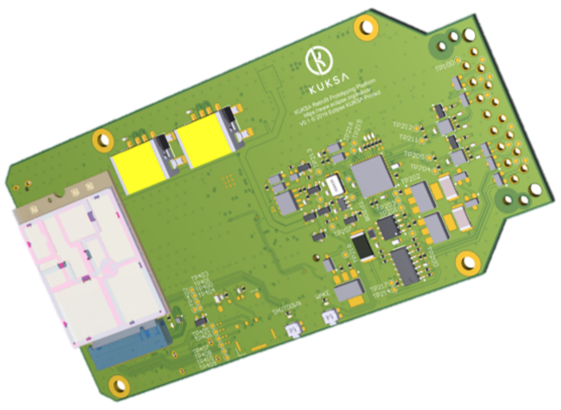


Figure 5: Schematic of the Kuksa Dongle. Equipped with a Raspberry Pi Compute Module, an OBD-II interface and CAN bus interfaces

The in-vehicular data processing pipeline is as shown in Figure 4. The pipeline starts by accessing raw in-vehicle data from field buses such as CAN using the OBD-II interface and the Eclipse Kuksa software stack. Kuksa is an open source vehicle data access framework jointly developed by Bosch. The software stack can be hosted on in-vehicle computing platforms like a vehicle computer for gathering the vehicular data from different sources (like on-board diagnostic interfaces such as OBD-II, in-vehicle field bus technologies such as Controller Area Networks, etc.) or on a retrofit solution based on customized hardware (cf. Kuksa dongle in the Figure 5). The collated data is made available over a web server hosted on the Kuksa stack. Eclipse Kuksa abstracts the vehicle specific details, thereby, providing a vendor-independent and vehicle-independent interface to the applications seeking to extract vehicular data.

Given that the landscape of automotive standards addressing in-vehicular data is fragmented, there are multiple standards which cover this space, e.g, the Genivi/W3C VSS [4], or the Sensor Interface Specification (Sensoris) [5]. Eclipse Kuksa uses the VSS specification, i.e., maps the in-vehicle data points to the VSS data model. Hence, for this PoC, we set up the data processing pipeline based on the VSS model.

The VSS models automotive data points, for example data from applications hosted on in-vehicle electronic control units, as a tree (cf. Figure 6) with a single root (Vehicle) and the data points (e.g., attributes, sensors, and actuators) as the leaves. Each data point can be addressed by traversing the tree from the root to the corresponding leaf. The leaf nodes could represent static data (data which never changes e.g. vehicle identification number), as well as dynamic data (sensor values or actuator inputs).

To the question of the transport protocol for this access, Eclipse Kuksa implements the W3C Vehicle Information Service Specification (VISS), an open standard to access VSS data over WebSockets and HTTP/IP. In addition to the traditional HTTP/IP request/response communication model, WebSockets allows Kuksa to communicate asynchronously using a bi-directional communication channel to deliver data to the applications, e.g., when there is a change in any attribute of interest.

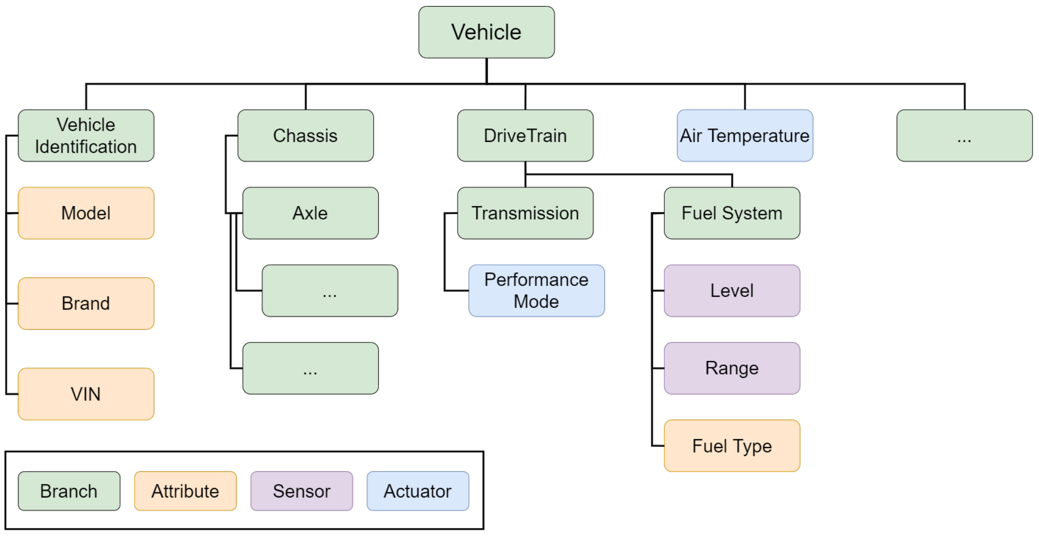


Figure 6: Example of the Genivi/W3C Vehicle Signal Specification structure represented as a tree of properties to be accessed [4], e.g., by Eclipse Kuksa.

Extracting vehicular data from a particular vehicle is only a job half done. The data needs to be processed in order to extract useful information for the risk evaluation model. We use the IoTea, a software stack implementing complex-event processing (CEP), for converting simple events (raw data like the one based on VSS data model provided by Eclipse Kuksa) into compound events (information/insights required by the behavioral risk evaluation model) based on pre-defined rules.

While there are several frameworks for data processing available, we choose IoTea for this PoC as its tooling already provides adapters for Kuksa to source input events mapped to the VSS data model, and therefore, in-vehicle data.

Based on the provided information, we will reason which of the presented solutions are hosted where (in-vehicle or the Piccolo infrastructure) in the data processing pipeline:

Kuksa Dongle & Kuksa Software Stack: Kuksa dongle interfaces with in-vehicle systems (like OBD-II port, the CAN bus etc.) to gather in-vehicular data and make it available using the Kuksa software stack. For these reasons, we see these components as a part of the in-vehicle systems.

IoTea Platform: The IoTea framework provides a rather simplified actor-based programming model to describe the processing rules in the form of the so-called Talents. Talents are the basic programming unit of IoTea. Each talent is a unit of work and can interact with other Talents via events (incoming events and generating outgoing compound events for other Talents). Talents form a distributed operator network while the underlying IoTea platform (basically a pub/sub like communication substrate) orchestrates the flow of events between the talents implementing principles from complex-event processing and offers additional services (such as rule evaluation when to inform a Talent of an "interesting" event). IoTea provides several types of Talents ranging from simple ones to describe the business logic using boolean algebra to complex interacting rules (like map-reduce).

Talents are implemented using the IoTea SDK. It allows to implement Talents using different pro-

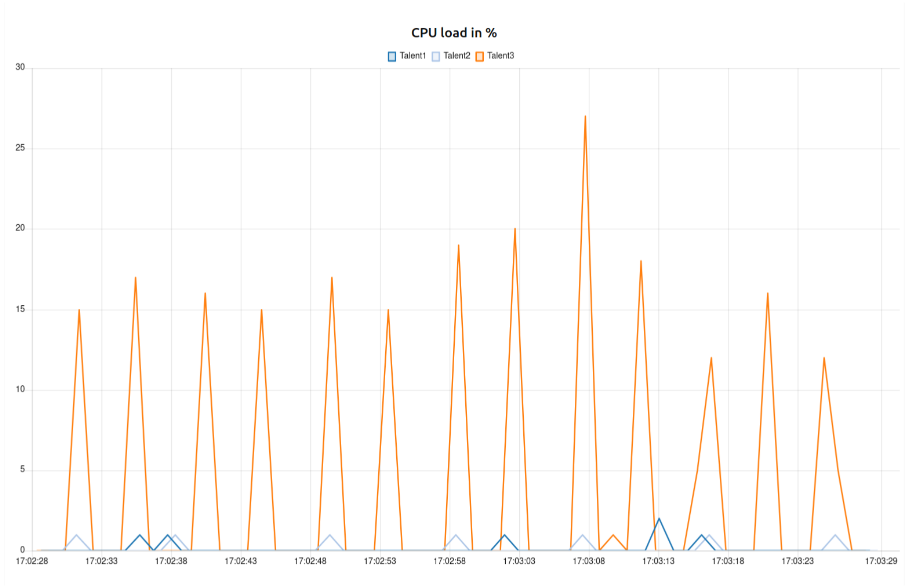


Figure 7: Results of a resource consumption benchmark of Genivi IoTea. IoTea Talents may generate high compute load, dependent on the business logic of the Talent, resulting in the situation that some Talents cannot be executed in-vehicle due to limited resources.

gramming languages (e.g., JavaScript, C++, or Python). Those Talents are able to be hosted on different machines being executed using the language specific runtime (e.g., C++, JavaScript or Python interpreter). Exchange of events/data is achieved via the IoTea platform using the publish/subscribe Message Queuing Telemetry Transport (MQTT) protocol. As a result, IoTea achieves a loosely coupled communication model, as Talents communicate independent of its execution runtime. This enables the execution of components already in a distributed fashion in different parts of the infrastructure as long as the required pub/sub substrate is available (here: at least one MQTT broker). For the purpose of this PoC, we intend to host the IoTea Platform (the MQTT broker) in the Kuksa Dongle (in-vehicle deployment), thus letting the data stream from Kuksa server to be filtered before forwarding only events of interest to the talents.

IoTea Talents for Implementing Business Logic: IoTea Talents are the processing units where the computations happen. Depending on the complexity of the rules and the load of incoming event stream, the computation load on individual talents can be beyond reach of what an in-vehicle platform can handle (cf. Figure 7, $Talent_3$). The pub/sub communication substrate of IoTea allows the Talents to be orchestrated across the infrastructure. For this PoC, we exploit this opportunity and distribute these talents within the Piccolo infrastructure using the Piccolo node API (see Piccolo Deliverable 2.1).

Trusted Execution of Data Processing Pipelines: Whilst our "Risk Monitoring" Proof-of-Concept helps to illuminate that there are new opportunities for services that build on in-vehicle and vision processed data, it also raises some privacy and security concerns. These include information about driving behaviour and the processing in the Piccolo network of raw (or semi-raw) data. This means that a trusted and secure execution environment in the Piccolo infrastructure must be used.

Technologies from the domain of trusted execution environments ensure that the data is processed by functions in a secure fashion, implementing the desired behavioural risk model and cannot be influenced by malicious code. A secure environment also ensures that the data (raw or processed) is secured against simple extractions and protected from use for purposes other than the evaluation of the behavioural risk.

For the PoC, we use the Trusted In-Network Computing (TINC) node from Fluentic Networks Ltd. to fulfill these security and privacy requirements. More information about TINC as part of the behavioural risk monitoring PoC is provided in Piccolo Deliverable 2.2.

3.1.3 Development Status

The Sensing Feeling Edge processing engine has been successfully connected to the Kuksa VISS client. This means that all of the data from Kuksa is provided by the Kuksa Dongle and accessible inside the Sensing Feeling processing pipeline, to send onto the Piccolo node or cloud for further calculations and analysis. In order to get access to the desired information required to form a risk behavioural model, a dedicated data model of vehicle data points is under development between the PoC partners based on the analysis of existing data models from the domain of vehicle insurance (e.g., Open Insurance Data Standard [1]), general models such as Genivi/W3C VSS as well as additional sources in the literature. Both developments form integral parts of this PoC, and proves that we can join the two data sets together in real time. Having these data points available on a single edge device makes it exceptionally easy to maintain, augmenting each other in a single data point that has both vehicle telemetry and visual processing telemetry. This also solves issues around Internet of Things identities, as all data can be transmitted via a single Edge device.

To be able to process in-vehicle data in a secure fashion, the preparation of VPE as well as IoTea to be executed on the Trusted In-Network Computing (TINC) platform is currently being integrated to a Piccolo node. While it is already possible to load and execute IoTea Talents dynamically on the TINC platform (in both, secure and not secure fashion), the preparation of the more complex VPE software is challenging. The reason is that the VPE has a lot of software dependencies and it needs to be re-compiled and linked against the TEE subsystem used by the TINC platform. Such a task is not trivial and requires certain modifications on the preparation process of the VPE for it to be able to run securely.

The specifications for the NVR kit have also been decided upon, including cameras and router.

3.1.4 Challenges

The model to determine behavioural risk currently needs further development. There are many factors that we know could affect such an index, but using these all simultaneously in a single balanced

[1] https://openinsurance.io/standards/](https://openinsurance.io/standards/

calculation is a bit trickier. While the vision sensing data is more experimental, the auto insurance industry has been using vehicle data (including in the forms of black boxes) to calculate risk for a much longer period of time. We can leverage this, using tried and tested methods, to identify what some of the key factors are from a vehicle subsystem standpoint. Notably, it's expected that vehicle speed, acceleration, turning are likely to be the main factors when calculating risk from the vehicle data. From the vision subsystem, key data points are expected to be a bit more varied. Pedestrians and vehicles in the field of view of the camera indicates there's more variance in the scene, with the risk of unexpected movement or other distractions. The attention of the driver itself will be a big factor, including any kind of eye drooping and signs of fatigue. Other occupants of the vehicle could factor in - if children are moving around on the back seats, for example, it could distract the driver further. Other external factors, such as lighting or weather, could also have an impact.

In terms of the hardware setup inside the vehicle, the identified issues are space and power. The cameras, along with the rest of the NVR kit, will need to be positioned such that they do not obstruct any fields of view and do not restrict any kind of movement inside the vehicle for the driver or passengers. Their positions will also need to be secured, so that there is no movement during general usage or in the case of a collision. Adding extra devices into the device will obviously have a power requirement. None of the suggested devices are battery powered, so would all need to be connected up to a sufficient power source.

Regarding the development of the Piccolo node as part of the communication infrastructure, there are several challenges to be solved. For example to support the IoTea platform. As IoTea uses MQTT as its transport protocol, it requires message broker instances to disseminate the data from data producers (vehicle) to the consumers (Talents or VPE in the network). However, the broker-based concept used by IoTea is challenging regarding the configuration and maintenance of communication routes in a dynamic service deployment, e.g., as supported by the TINC platform. One potential 'solution space' is the evaluation of location-independent servicing concepts from the domain of data-oriented networking.

3.1.5 Next Steps

The main next step is to decide upon the calculations required to give a measure of risk. Once the data points from the vehicle subsystem are decided upon, they must be reliably retrieved from the vehicle. Given the variance of vehicles, where hardware and inner systems can be drastically different, we need to avoid the chance that a specific vehicle can not provide the data the algorithm is dependent on.

Once this is done, the setup must be tested in a real vehicle. This includes calculating power usage, as mentioned above, as well as the placement itself. Viability of this solution is dependent on this. The edge device that we use for the image transmission to the node will also need to be decided, depending on power constraints and how many cameras are used in the setup.

3.2 Smart Factory

3.2.1 Context & Goals

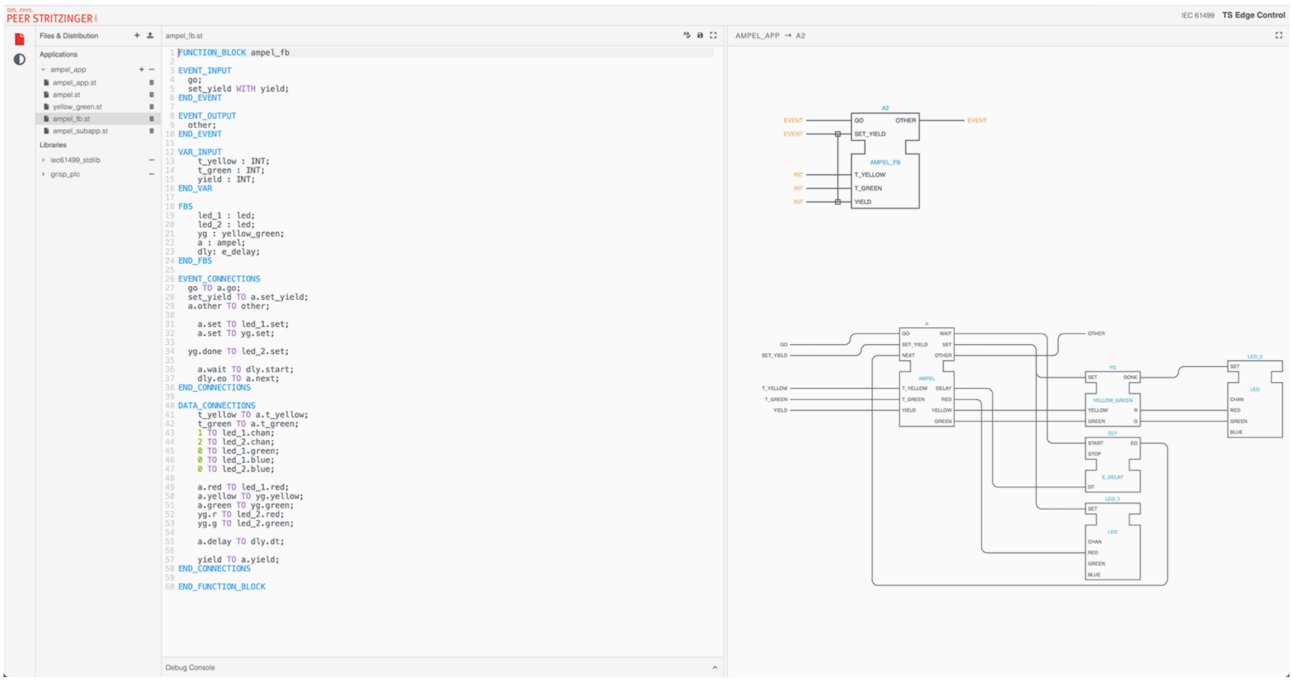
Transport systems in factories, which are responsible for moving partially completed products between processing stations (manual and automatic), are a crucial part of infrastructure. A very common one consists of a pair of narrow conveyor belts on rails with square workpiece carriers carried along via friction. There are several possibilities to stop, sense and move the workpiece carriers in different directions. Additionally there are inductive sensors giving a signal when a carrier comes by, mechanical switch sensors that detect a carrier along a whole range, and Radio-Frequency Identification (RFID) antennas that can be used to detect the presence of a carrier and to identify production metadata. The main elements of influencing material flow are so called separators consisting of a bolt that can be moved upwards between the conveyor and which results in the next carrier being stopped (the belts keeping moving, slipping). Stopped carriers can be queued before such a separator in a space efficient way. The underside of the carriers have a specific form that allows a separator to let through the first of a queue of carriers and stop the next by briefly opening. For workpiece carriers to move in different directions there are Lift Transverse Units (in German Hub-Quer or short HQ). They can be moved vertically into 3 positions: on the lower position carriers just move straight on over it; on the middle position one carrier can be stopped and when detected lifted to the upper position which lifts it off the main conveyor and moves it onto a orthogonal conveyor. The same can be used for merging carriers onto the main conveyor.

We outline the state of the art for controlling the material flow along assembly lines. First mechanical design and the setup of the conveyor parts, sensors and actuators, which are mounted and connected to a central Programmable Logic Controller (PLC). After that the material flow has to be programmed individually for each assembly line in a PLC compatible language (e.g. IEC 61131 [6], see also figure 8). Due to the limited capability of PLC languages material flow is usually programmed quite statically, which often creates a limitation on throughput. Programming the PLC and also reprogramming on changes of the assembly line is a considerable repeated effort.

The overall goal is to develop a new kind of smart conveyor belt consisting of macro modules each with its own small controller nodes. These controller nodes are already connected to local sensors, RFID antennas and actuators and will be networked to neighbouring nodes. After the setup of such a conveyor belt topology it self-learns the mechanical topology and provides "Plug&Produce" routing functionality. This modular approach with many small nodes necessitates a fully distributed orchestration of distributed computation. The advantage is that the computation capacity scales automatically with the size and complexity of the assembly line.

3.2.1.1 Expected efficiency gains:

- Improved machine utilisation and allocation.



```
FUNCTION_BLOCK ampel_fb

EVENT_INPUT
  go;
  set_yield WITH yield;
END_EVENT

EVENT_OUTPUT
  other;
END_EVENT

VAR_INPUT
    t_yellow : INT;
    t_green : INT;
    yield : INT;
END_VAR

FBS
    led_1 : led;
    led_2 : led;
    yg : yellow_green;
    a : ampel;
    dly: e_delay;
END_FBS

EVENT_CONNECTIONS
  go TO a.go;
  set_yield TO a.set_yield;
  a.other TO other;

    a.set TO led_1.set;
    a.set TO yg.set;

  yg.done TO led_2.set;

    a.wait TO dly.start;
    dly.eo TO a.next;
END_CONNECTIONS

DATA_CONNECTIONS
    t_yellow TO a.t_yellow;
    t_green TO a.t_green;
    1 TO led_1.chan;
    2 TO led_2.chan;
    0 TO led_1.green;
    0 TO led_1.blue;
    0 TO led_2.blue;

    a.red TO led_1.red;
    a.yellow TO yg.yellow;
    a.green TO yg.green;
    yg.r TO led_2.red;
    yg.g TO led_2.green;

    a.delay TO dly.dt;

    yield TO a.yield;
END_CONNECTIONS

END_FUNCTION_BLOCK
```

Figure 8: IEC61499 IDE

- Flexible adaptation to varying machine efficiency and outages.
- Retrofitting to other product type with less or no downtime.
- Minimizing the programming effort and downtime on new assembly lines and on changes (which are frequent enough that this makes a difference).
- Optimizing mechanical topology by being able to simulate before building.
- Optimizing number of carriers on a line (state of the art is applying rough rule of thumb).

When combining all the effects above, efficiency gains of up to 40% are the target goal, depending on how far an existing solution is from the optimum.

3.2.2 Architecture & Design

Since the conveyor belt system comes with its own compute nodes there should be no necessity for other nodes besides the small Internet of Things (IoT) nodes. Still, computation power could be added optionally and 5G edge nodes or cloud nodes could be used without changes to the distributed software stack.

The Industrial GRiSP nodes run a unikernel based on the GRiSP software stack which consists of an Erlang VM directly running on hardware via the use of Real-Time Executive for Multiprocessor Systems (RTEMS). The hardware will consist of a NXP i.MX6UL embedded CPU with ARM architecture (see figure 9), two 100Mbit/s Ethernet connectors, one RFID antenna and 8 configurable

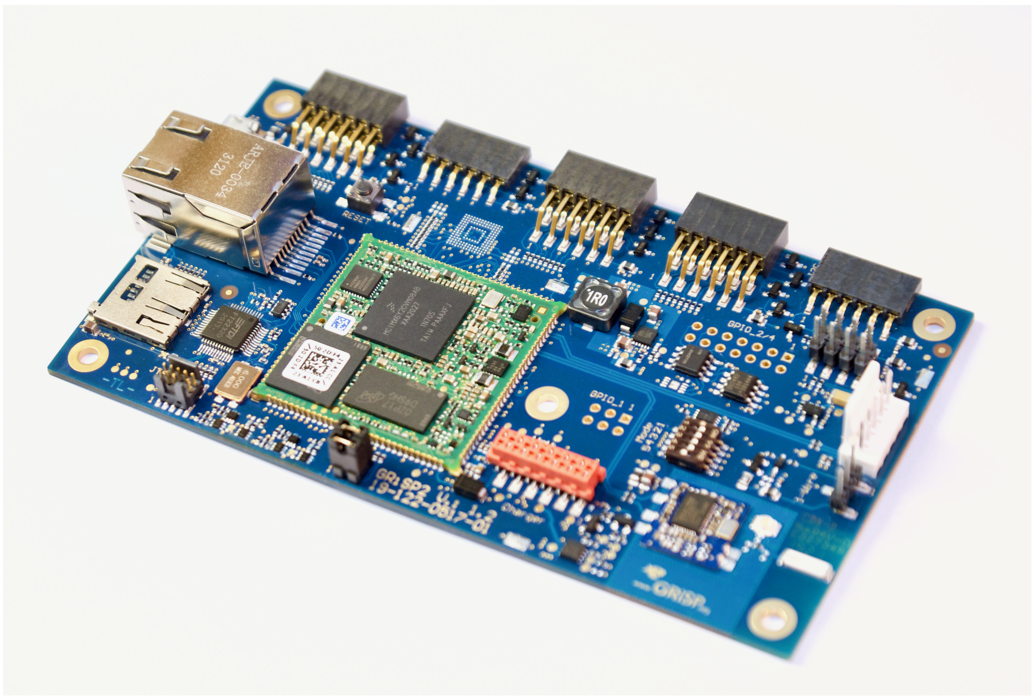


Figure 9: GRiSP2 Evaluation board

industrial 24V Input Outputs (I/Os).

These embedded nodes will be connected with neighbouring nodes via Ethernet and optionally via switches to form a mesh-network of nodes.

On these embedded nodes an instance of the unikernel (which contains the Erlang VM) is installed together with the application written in Erlang and IEC 61499 [7] Structured Text (a distributed PLC programming language which is compiled to the object files of the Erlang VM to run seamlessly with the Erlang code). With the help of RTEMS (a library OS) the Erlang VM runs directly on hardware, allowing for smaller and more energy efficient hardware, providing soft real time performance with a tie-in hard real time if necessary (see Figure 12).

The Erlang VMs are connected with neighboring node Erlang VMs and use Erlang's built-in transparent distribution protocol to provide a uniform execution model on heterogeneous edge compute node up through all intermediate layers and the cloud if necessary.

The software architecture consists of three larger subsystems:

Distributed low level control loops to control a subsystem of the conveyor belt system like a queuing area in front of a manufacturing process or an intersection The control loops connect to sensors and actuators within a network and mechanical locality providing higher level actions like "move the next carrier left at this intersection". Integrated with the lower level control system is a mechanical topology learning module that learns automatically the mechanical topology after changes by running empty carriers repeatedly through the paths. During the time the mechanical

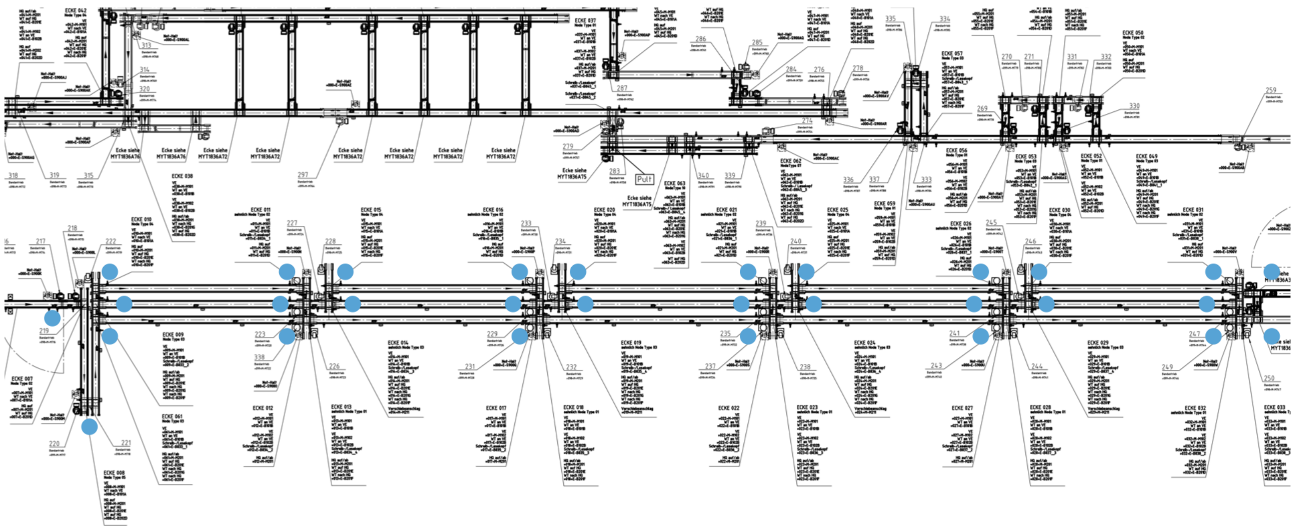


Figure 10: Nodes located along an assembly line

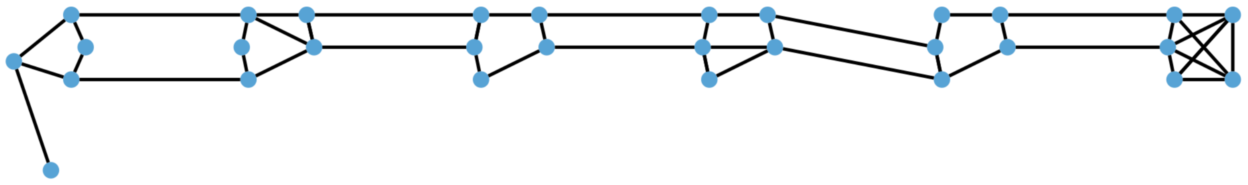


Figure 11: Mesh network with neighboring nodes connected via Ethernet

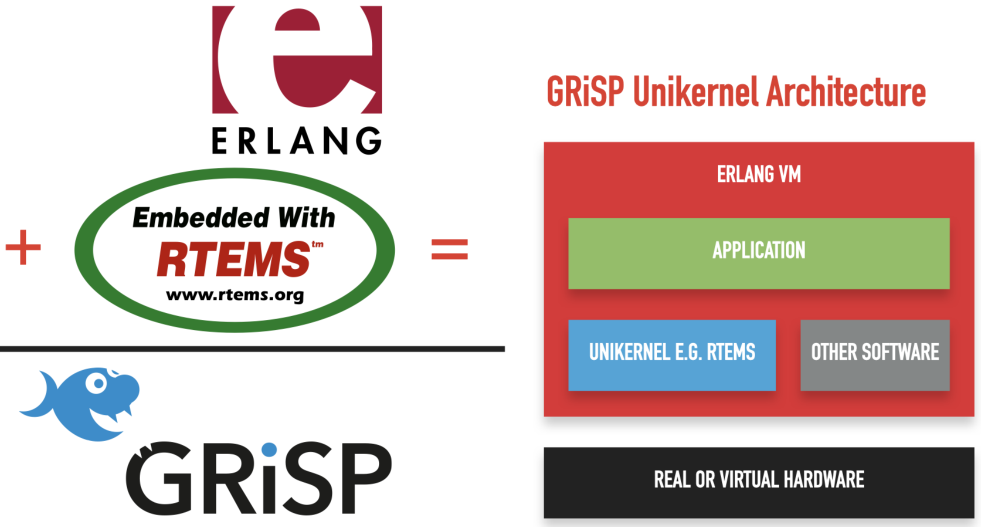


Figure 12: GRiSP Architecture

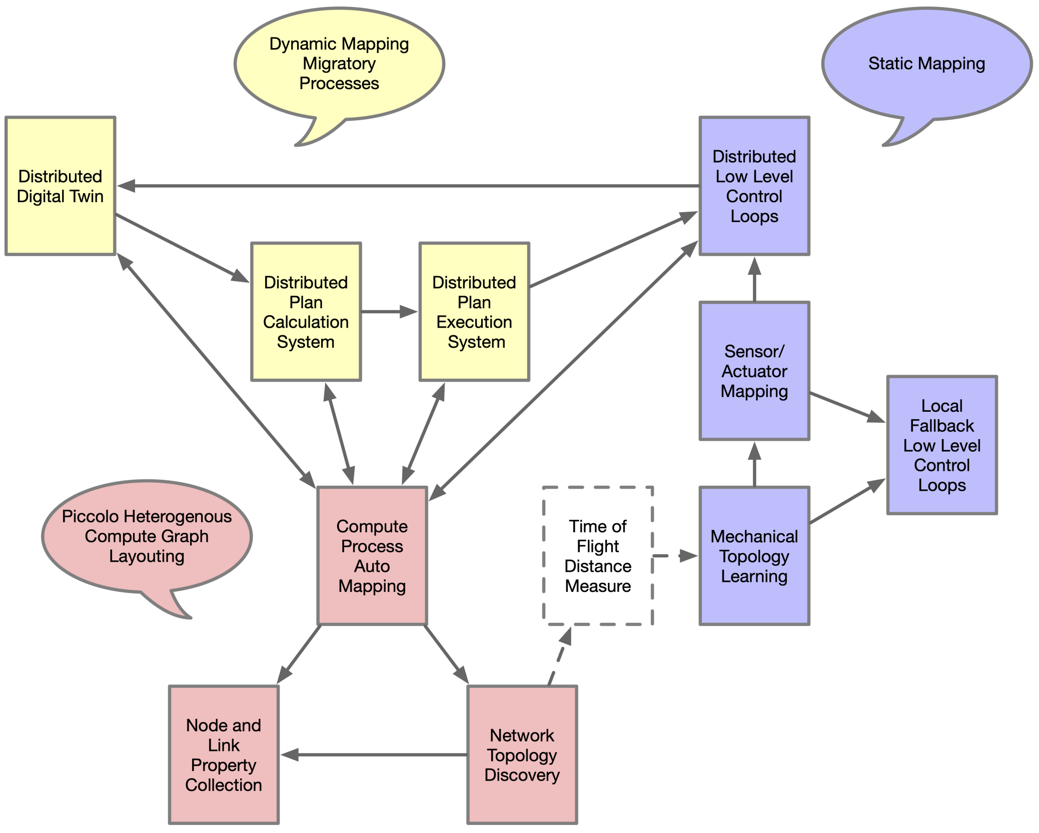

Dynamic Mapping Migratory Processes
Static Mapping
Distributed Digital Twin
Distributed Low Level Control Loops
Distributed Plan Calculation System
Distributed Plan Execution System
Sensor/ Actuator Mapping
Local Fallback Low Level Control Loops
Piccolo Heterogenous Compute Graph Layouting
Compute Process Auto Mapping
Time of Flight Distance Measure
Mechanical Topology Learning
Node and Link Property Collection
Network Topology Discovery

Figure 13: Architecture of distributed Plug&Produce transport system

topology is unknown or on network failures a local fallback control system can provide sub-optimal but safe operations (blue colour blocks in Figure 13).

Distributed implementation of a digital twin of the state of conveyor belt equipment together with the positions and metadata of all carriers Since we can only gather information when a carrier passes by a sensor or RFID antenna this gives us a very sparse view of the overall state. This digital twin fills in the gaps and gives us a best approximation simulation of the real world. This simulated state is used as a starting point for a planning system which tries to find a close to optimal solution for material distribution. The calculated plan should avoid carrier traffic jams and provide optimal and balanced (to avoid uneven wearing out) machine usage and throughput. This computation will be running in a distributed way to make use of all the compute power of the numerous embedded nodes allowing a standalone operation of a smart conveyor belt system. These calculated plans are passed to a plan execution system that uses the low level control layer. The digital-twin together with the plan execution subsystem will detect deviations from the planned sequences and re-trigger the plan calculation from the currently known state. This altogether implements a built-in distributed online planning system (yellow colour blocks in Figure 13).

Topology learning To map all the distributed compute processes of the other subsystems, the network topology needs to be learnt either by accessing routing protocol data (e.g. IS-IS or Shortest-

Path-Bridging) or by running a link state protocol on top of Erlang's distribution protocol. Together with these link state packets, node and link property data (like capacity, sensor and actuator location) can be learned by the whole network converging to a common view of the network. This information can be used to provide a distributed implementation of automatic mapping of processes to compute nodes, according to constraints, mainly to balance load optimization with low latency (red colour blocks in Figure 13).

3.2.3 Development Status

Together with the use-case partner Bosch Bühl, one of the electric motor assembly lines to implement the use-case was identified and the necessary stake holder buy in has been achieved. This electric motor assembly line is currently in full production and already has a spooling machine pool consisting of five spooling machines each with a capacity of six synchronously spooled motors. The spooling process is the slowest process in the production sequence and therefore built with high concurrency capability. We will focus on the material distribution across this spooling pool since it has the highest potential for optimisation.

A fully distributed product built into the assembly line is the goal for new greenfield projects and also as part of a new smart conveyor belt product line of the manufacturer Bosch Rexroth. However it is very important to improve brownfield projects which are already fully functional. In this use case we have such a brownfield project, since the assembly line is already fully functional and has been operating for several years.

Together with the use case partner a risk mitigation strategy has been developed to allow seamless switching between legacy systems and the new to be developed controller. The legacy system consists of a central PLC controlling the whole conveyor belt material flow and all connected sensors, actuators and RFID subsystems. As a first step a pure software integration to this PLC is implemented to provide access to all sensor data and allows to control all actuators via this PLC. After specification of an OPC Unified Architecture (OPCUA) based data exchange scheme we have implemented the necessary features on top of our existing OPCUA stack (implemented in Erlang, already existing intellectual property). Implementation of a data sharing and control switching layer in the PLC has begun by the use case partner. This will allow us to start software implementation before the Industrial GRiSP hardware is available. At the same time we are developing the necessary drivers for Industrial GRiSP with GRiSP2 evaluation boards which have been developed beforehand and Pmod (Peripheral module interface) modules for RFID and Industrial Digital 24V I/O the hardware for which have been developed by us outside of Piccolo funding.

Development of a simulation subsystem has been begun which allows us to simulate conveyor belts with carriers and the respective actuators and sensors. This simulator in Erlang, which will be developed in a distributed way, will be used multiple times in the future:

- For developing the complete system without constant access to a conveyor belt system with a variety of mechanical topologies not available otherwise.

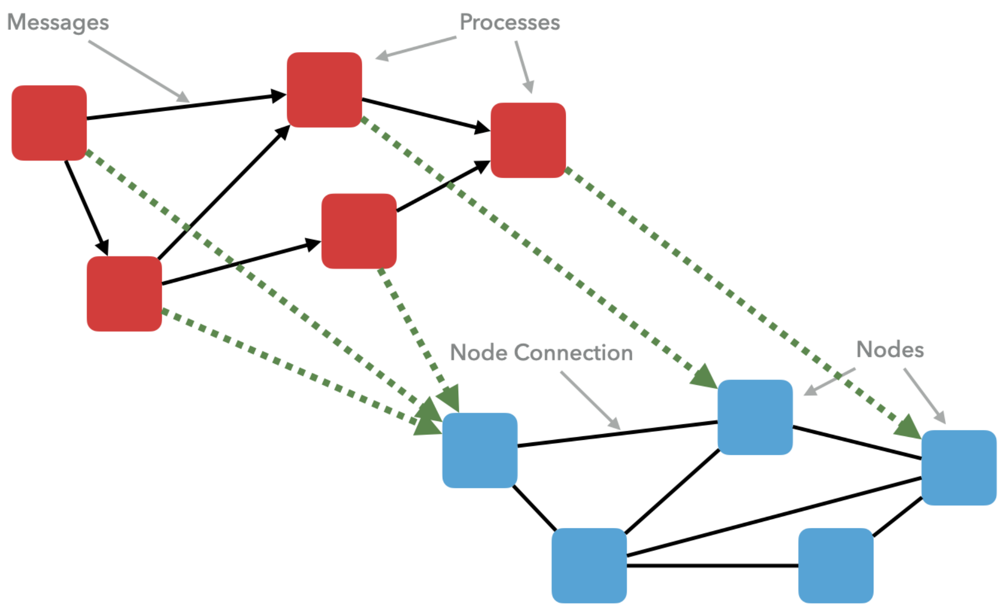


Figure 14: Mapping Message Passing Processes to Nodes

- For visualisation of the conveyor belt with all carrier positions at the use case partner (relieving a current pain point at the use case partner).
- As simulation component for the distributed digital twin, which converges towards reality by repeated sensor input and eventually providing a complete starting state for the planning algorithms.
- As a pre-sales support tool for Bosch Rexroth to allow their customers to plan and simulate conveyor belt system before ordering the parts.

3.2.4 Challenges

The overall challenge is to build a complex distributed system that integrates all the functionality mentioned above: mechanical topology learning, distributed digital twin, distributed plan calculation and execution system. Having all these subsystems implemented and distributed is the prerequisite to having a really smart industrial transport system with embedded intelligence. If the influence of network properties on performance is too large then intelligent placement of functionality can't fix this. The distributed nature of the sensor/actuator connections together with the limited power nodes prevents the concentration of all computation on one node.

Because of the arbitrary network topology and the Plug&Produce requirement, manual function placement is precluded so the next challenge is automated function placement (see figure 14).

3.2.5 Next Steps

Design for the above mentioned simulator for transport systems has been finished and implementation has begun.

3.2.5.1 Simulator Workpiece carriers are square and systems exist in a limited number of track width from 80 mm x 80 mm up to 1.200 mm x 1.200 mm.

We model the carriers as basically circles with a radius r and centre moving along the middle of the conveyor belt tracks. Since mainly orthogonal tracks are covered the circles behave identical to the square carriers. For visualisation we show squares and along the conveyors there are sensors detecting the presence of a carrier, these are only activated normally in a sub-region of the carrier where there is a metal area underneath it.

There are also separators, which are actuator bolts coming up from below and stop the next carrier at the front. Before such separators carriers can collide and queue up. Opening the separator for a short time and then closing it again lets exactly one of the queued carriers through (there is a cutout underneath that allows the bolt closing while the carrier is still moving over it which allows to reliably stop the next carrier again). The carriers are moved along with constantly moving ribbons on the sides so when they are stopped the ribbon glides under it with friction. When moving there is no relative movement between ribbon and carrier. Effects of the conveyor slowing down and friction times until carriers are up to speed are ignored in the simulation for now.

There are main conveyors and transverse conveyors, usually running at 90° to each other. To get from the main conveyor to a transverse conveyor there are Lift Transverse Units (HQ). They have three positions: 'Down' means carriers can pass over it and keep going on the main conveyor. 'Middle' stops the carrier on the unit, there is a special sensor attached to it that signals the presence of a carrier to the control system. 'Up' starts moving the carrier in the direction of the transverse conveyor. In a Lift Transverse Unit only one carrier is allowed, so usually there are separators in front of them. Lift Transfer Units can also be used to have carriers coming on the transverse conveyor and insert them into a main conveyor. Often there are loops of main conveyors with transverse conveyors at the ends and at certain points in the middle to have carriers go in circles. There can be also several parallel main conveyors connected by transverse ones to allow overtaking etc.

In addition to the normal sensors there are also RFID antennas which allow read/write access to the RFID tags mounted in one corner of the carrier. When stopped one can keep reading and writing to the tag (most common case). There is also the possibility to read a limited amount of data in drive-by mode. Most of the time data is not written then for reliability reasons.

3.2.5.2 Architecture A lot of thought has been given to the architecture of this simulator and different approaches explored. For sure it will be event based and not step based i.e. the simulation calculates for each element when the next event like stopping or passing a sensor happens and then

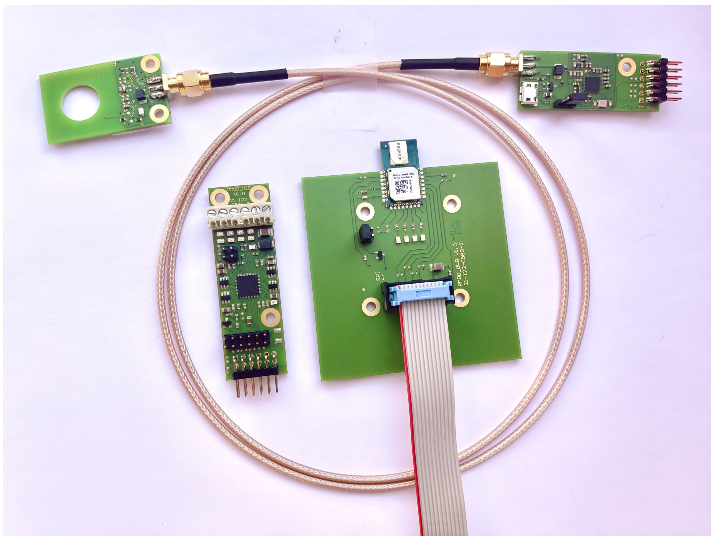


Figure 15: PMOD RFID, PMOD DIO and PMOD UWB

either uses a timer in the real time case or skips forward to the next event without waiting. This is so simulations can run as fast as possible. For distribution and implementation in Erlang a fair number of concurrent Erlang processes are required. This incidentally also greatly speeds up performance on large servers with many cores, which the Erlang VM manages very efficiently in case enough concurrency exists in the implementation. Avoiding synchronisation of concurrent parts is also crucial for good performance in both cases. First architectures with a controlling process per conveyor belt segment either with the carriers as state or as separate processes have been explored. The currently favoured architecture however consists of processes for sensors and actuators (which also have controlling processes in the real world implementation of the control system) and a process for each carrier. The conveyor belt stretches are passive data since they usually won't change. Most of the time the conveyor belts keep running at constant speed and only get switched off to save energy in phases of inactivity which doesn't need to be part of the simulation. Queues of carriers are kept in the state of the actuator processes of the separators.

3.2.5.3 Further next steps Further next steps are connecting to the legacy PLC via OPCUA and collecting complete data from actual manufacturing. This tests the connectivity and provides us with data to develop the digital twin synchronisation offline without access to the assembly line. Prototype hardware developing is ongoing, adapting our GRiSP software stack to these prototypes.

For developing device drivers and adapting the application before availability of Industrial GRiSP hardware we have developed PMODs modules to be plugged into the GRISP2 board (Figure 9). These PMODs have been funded outside of Piccolo. There is one PMOD for 15MHz RFID standards including a antenna. Another one for four 24V PLC compatible Digital I/O which is cascadable to allow up to 16 I/Os on one PMOD connector (there will be the equivalent of 2 PMODs on the Industrial GRiSP hardware). Finally a PMOD for Ultra Wideband 7GHz 802.15.4a radio (Figure 15. The latter will not be part of industrial GRiSP but used for exploratory work towards the end of the project to integrate time of flight location triangulation and a wireless data layer to the capabilities.

3.3 In-Network Hardware Acceleration for Time-Sensitive Networking

Time-Sensitive Networking (TSN) is layer 2 based Ethernet enhancement defined by IEEE 802. From its origin in 1973 Ethernet technology was stochastic. Over a shared transmission medium packet collisions and retransmissions with random waiting times were part of the concept. The technology was called carrier-sense multiple access/ collision detect (CSMA/CD). Later, in 1989 the introduction of switched Ethernet eliminated the random waiting times and resolved the collisions by queuing inside the switches. In 1995 the prioritisation of frames was introduced to reduce the queuing times for real-time traffic. Later, in 2012 the work on TSN started with the goal to eliminate the stochastic nature of frame multiplexing by reserving time slots for frames.

Time slots were known from transmission lines in telecommunication networks such as PCM30/ E1 and the synchronous digital hierarchy (SDH). Basic requirements for these technologies are synchronised clocks and a repeating frame structure. The smallest unit in these technologies is the 64kbit/s voice time slot. In TSN the slots have configurable size of at least one Ethernet frame, with minimum 64 Byte. Clock synchronisation is achieved by sending frames with precise time stamps between nodes, known as Precision Time Protocol (PTP).

PTP specifies transmission of sync frames containing, for example, the transmission time of a previously sent sync frame with a nanosecond accuracy. Such precision requires hardware support. The same applies to the frame structure and the time slots. For example, a 3-port Ethernet switch could have two TSN ports and one legacy Ethernet port as shown in Figure 16.

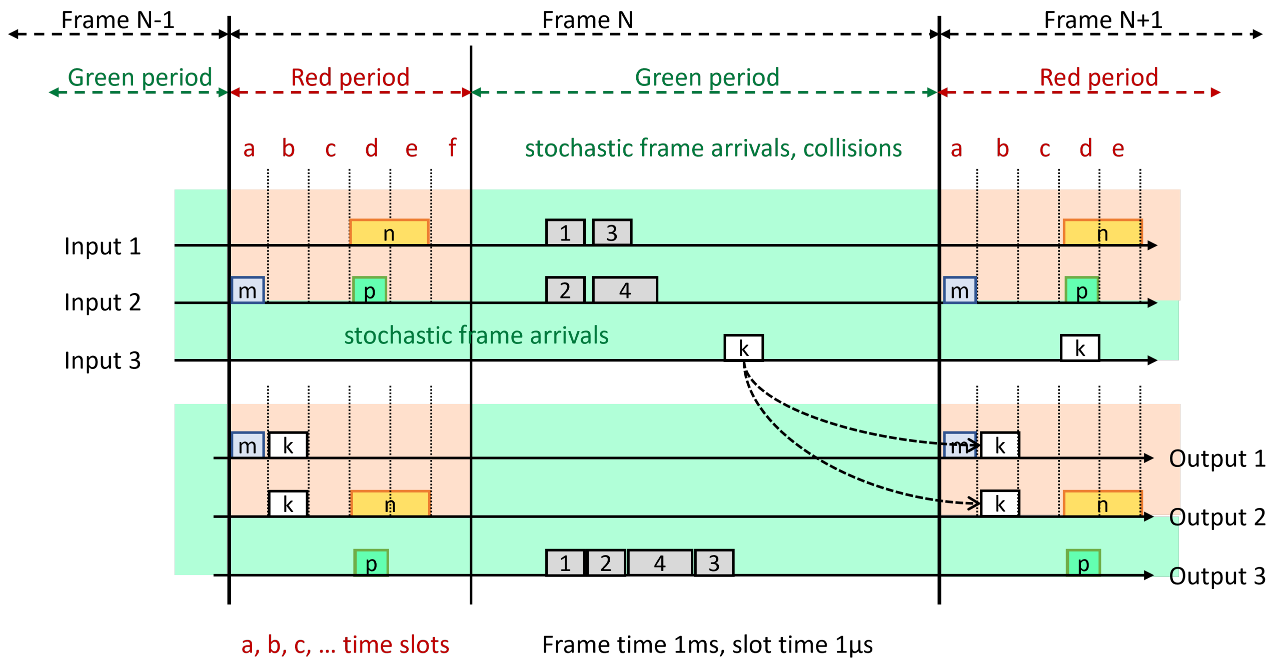


Figure 16: Example 3-Port TSN Switch

Figure 16 shows several facts about TSN:

- TSN links (Port 1 and 2) have a frame structure with a slotted period (shown in red) and a statistical multiplexing period (shown in green). Non-TSN links (Port 3) are legacy Ethernet type. The values for the frame time and slot time are exemplary.
- The time slots in the red period are reserved for periodically arriving frames (named n, m, p). These frames are not buffered, but forwarded immediately with (small) processing delay only. Note that periodic frames typically occur with analog-to-digital converters.
- In this example a slot time of 1000ns has been chosen. Assuming a Gigabit Ethernet link this leads to a slot size of 125 byte. For larger frames 2 or more slots are reserved (e.g. frame n).
- Within the green period frame arrival is stochastic and hence frames dedicated to the same output can arrive simultaneously and must be buffered.
- Frames k arrive periodically, but with jitter, at the non-TSN input 3, but are forwarded in a reserved time slot (b). They are duplicated to outputs 1 and 2. As time slot (a) is already occupied for output 1, slot (b) is selected for both. This feature could be used for the PoC in Figure 23.

Obviously, the configuration of a TSN switch is quite complex. Frames and time slot must be configured and for each flow slots must be reserved and aligned with other switches in a network. This breaks with the Ethernet paradigm of stateless packet sending.

3.3.1 Context & Goals

TSN is often mentioned as enabling technology for the convergence of Information Technology (IT) and Operational Technology (OT) . The benefit of this convergence is visualised along with Figure 17 and Figure 18. Today, the IT world terminates at an adapter which controls a specialised real-time bus at the OT side (Figure 17). With the real-time capable Ethernet enhancement TSN the IT can directly control a sensor at the OT side (Figure 18). However, this direct control comes at the expense of complex configuration (control and management plane). The Piccolo Agent concept could be a solution to do this configuration as an in-network function (Figure 19).

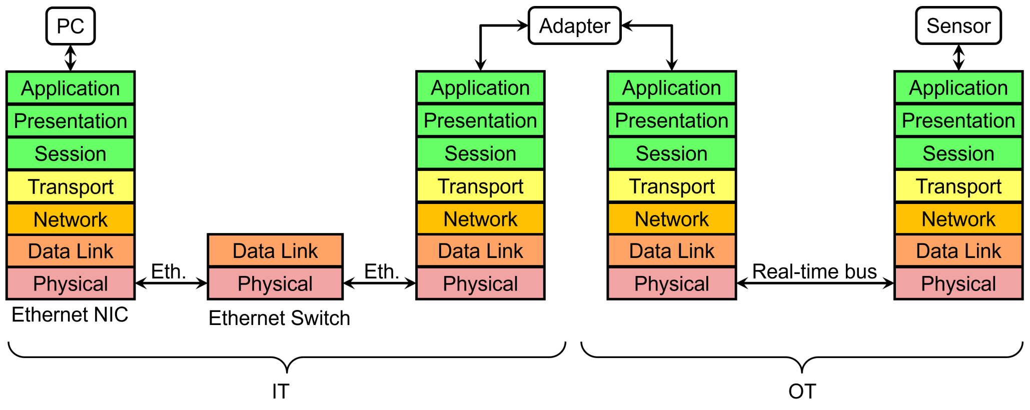


Figure 17: Complete separation of IT and OT today

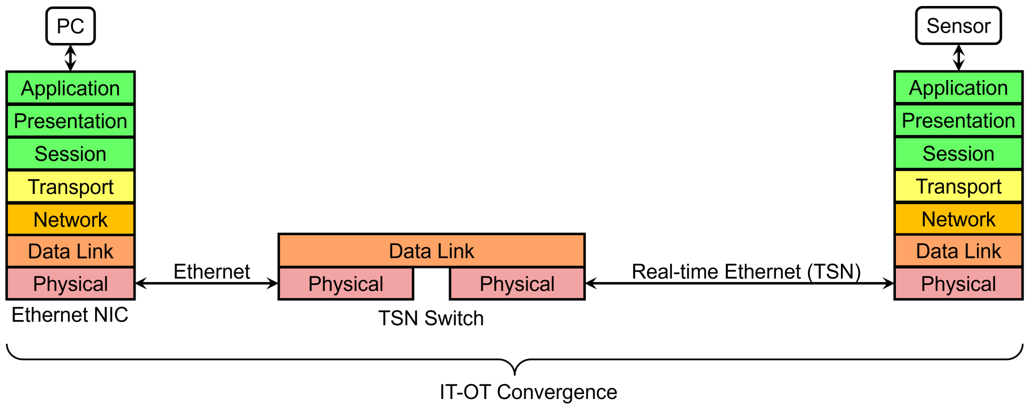


Figure 18: IT-OT Convergence allows seamless access to devices

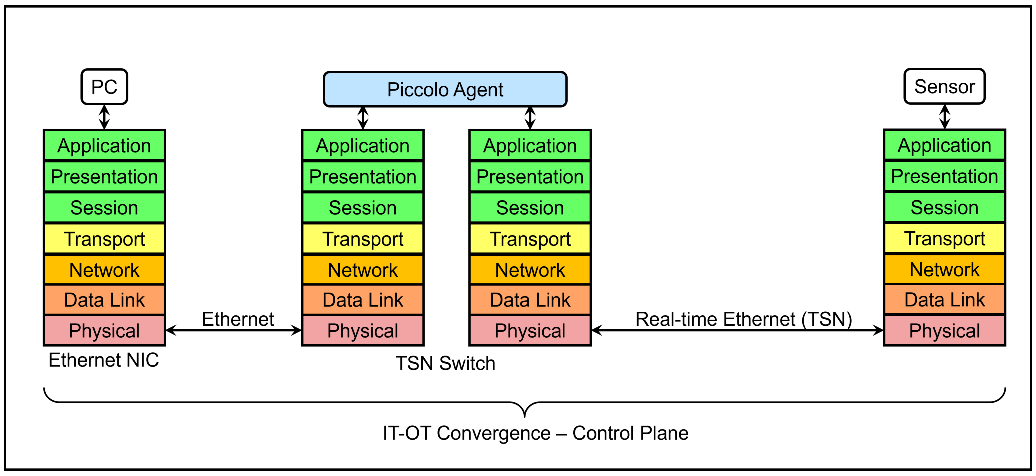


Figure 19: Control and Management plane solved by Piccolo Agent

Figure 18 shows that the control plane of the TSN switch terminates configuration messages from both network side and sensor. The Piccolo Agent block symbolises an in-network functionality to be investigated.

3.3.1.1 Problem statement

The generic Piccolo node, as defined in D2.1 [8], consists of 'default' computing processor supported by specialised hardware acceleration as shown in Figure 20.

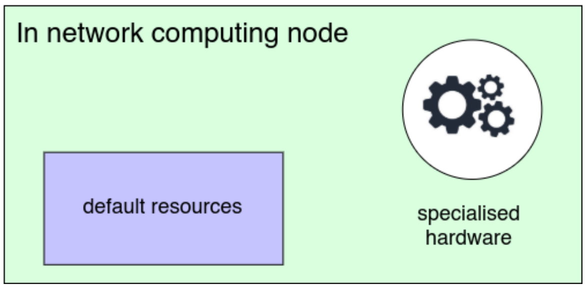


Figure 20: In network computing node with specialised hardware extension

Networking today is mainly best effort data forwarding, in layer 2 with Ethernet switching and in layer 3 for Internet traffic.

Transition from best effort to guaranteed quality of service (QoS) is a major step involving data and control plane processing. In particular TSN is an area where complex control protocols and dedicated hardware are required for on-the-fly processing of data-streams. Therefore, InnoRoute proposes to investigate the option to introduce QoS and TSN as in-network computing.

3.3.1.2 Potentials for in-network compute

TSN technology includes a series of IEEE specifications of the 802 series completed by NETCONF, RESTCONF, and OPC UA or YANG data structures as described in [9]. Page 5 of this reference describes a fully centralised model for flow configuration in TSN networks and is copied here in Figure 21.

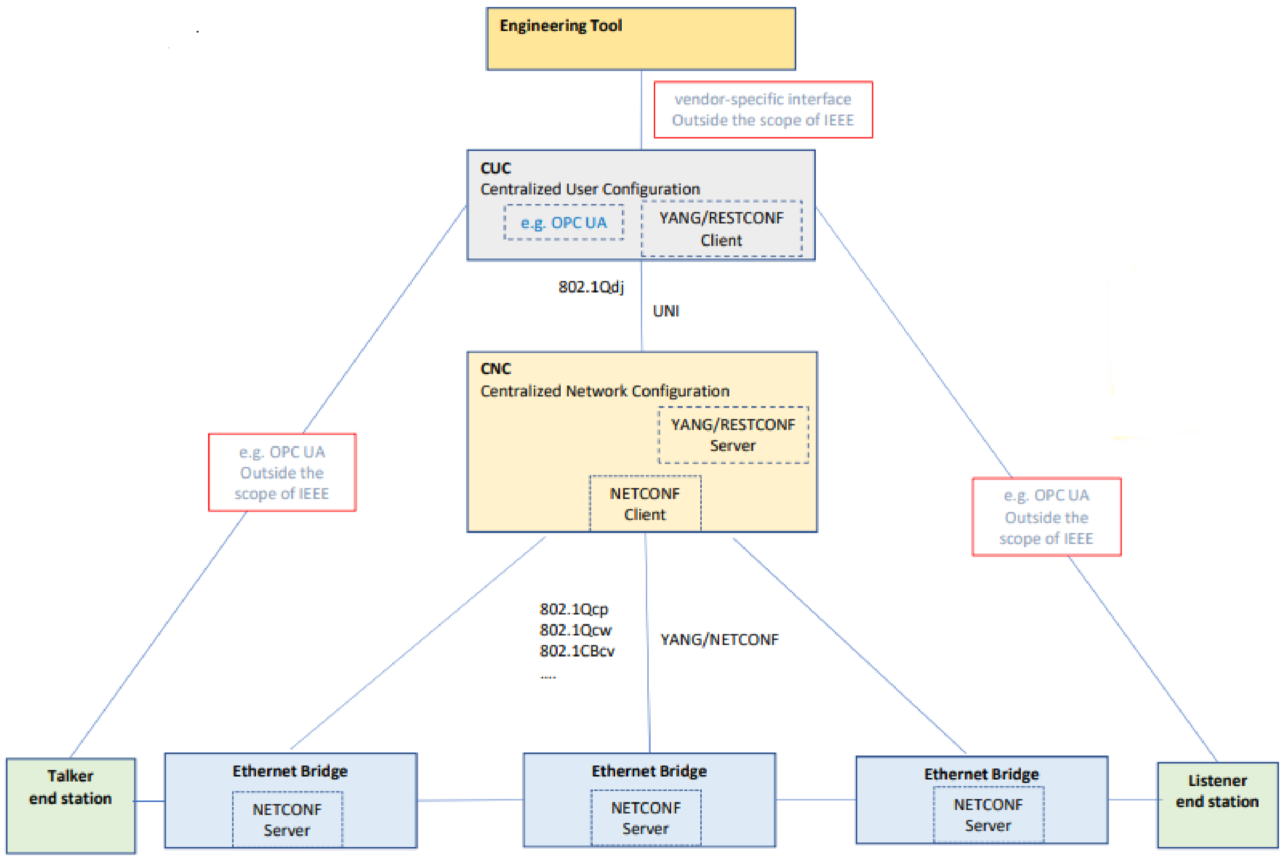


Figure 21: Fully centralised model for TSN flow configuration

In Figure 21 red rectangles indicate interfaces which are not covered by IEEE. All other components and interfaces are specified and will be available in future TSN switches and control software packages. Within this PoC the "Engineering Tool" will be considered, which obviously could be the Piccolo Agent.

The integration of specialised network hardware into in-network compute enables networks to benefit from in-network compute advantages. Figure 22 shows this convergence between a classical in-network compute setup and an Industry 4.0 (I4.0) network. The I4.0 environment (red) is included and controlled from default in-network compute entities (blue), the translation between both domain is realized by a hybrid node (green) which encapsulates standard I4.0 devices from the control network.

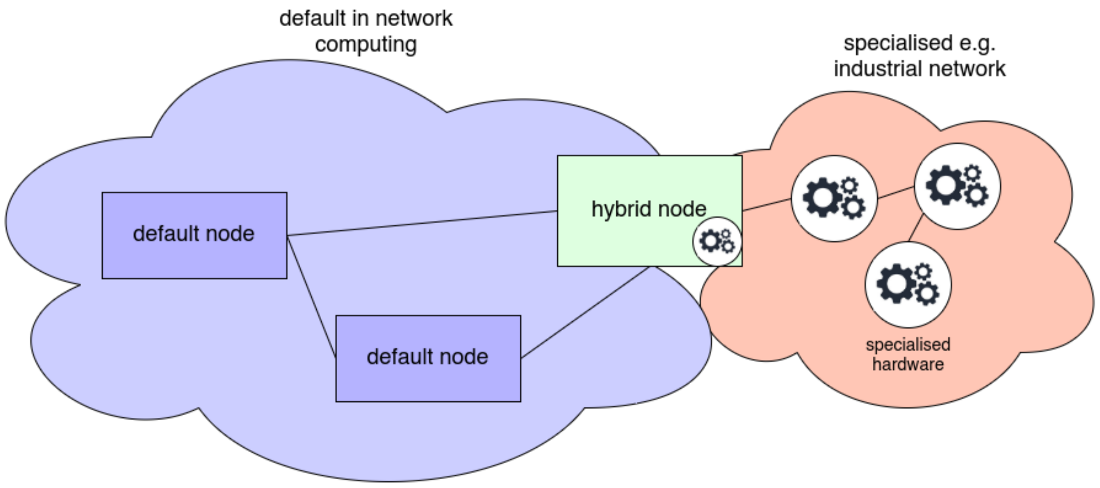

default in network computing
specialised e.g. industrial network
default node
hybrid node
default node
specialised hardware

Figure 22: In network hybrid usecase, involve I4.0-nodes

3.3.1.3 Key requirements & Challenges

The key challenges of this PoC implementations are mainly the interfaces which are not specified by the standards (red blocks in Figure 21):

- Development of lightweight Central User Configuration and Central Network Configuration functionality.
- Adaptation of Piccolo Agent as "Engineering Tool"
- Specification of interface between Centralised User Configuration and Talker/ Listener end stations as Piccolo interfaces.

3.3.1.4 Stakeholders

- I4.0 and TSN network providers and factories
- TSN switch manufacturers

3.3.2 Architecture & Design

The goal of this PoC is not a specific use case, but mainly to demonstrate the use of in-network computing for TSN node configuration. Therefore, a use case has been selected which allows the demonstration with minimum effort.

This PoC implementation demonstrates the integration of a highly time critical industrial process as a dynamical loaded function into the Piccolo ecosystem. The basics of industrial communication based on OPCUA are described in Del2.1 [8]. It is based on the basic docker node, described in Del2.2 [10] and the TrustNode routing platform. Additionally a customized RaspberryPI, which we term RealTimePI, is used to realise the setup. The PoC setup will address the key challenges from Section 3.3.1.3, by integrating the I4.0 setup into Piccolo, by utilising Piccolo components and interfaces. The existing I4.0 interfaces shown in Figure 21 will be mapped to Piccolo functions and interfaces to allow other Piccolo components the configuration of the I4.0 subsystem. The I4.0 subsystem is represented by abstract components as shown in Figure 23.

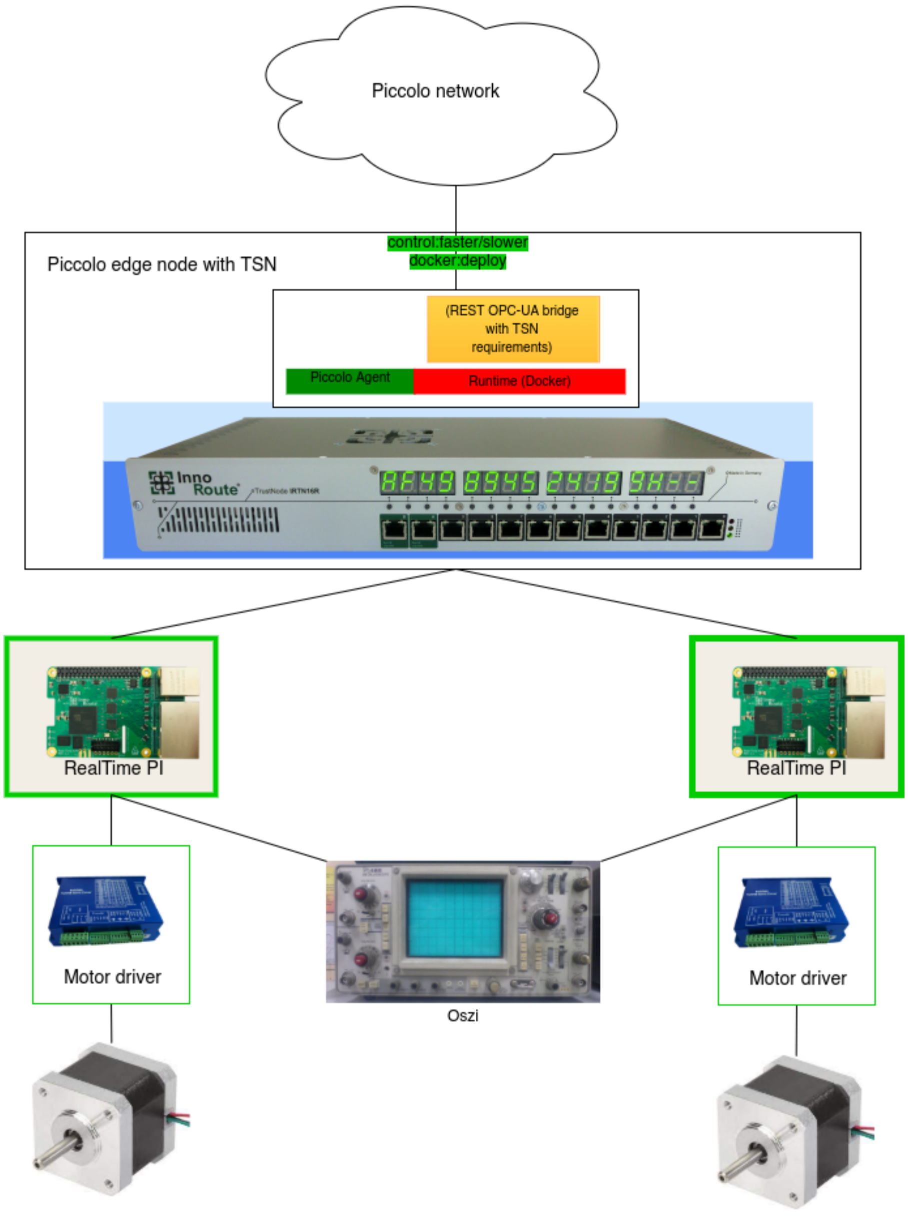


Figure 23: Piccolo docker - OPC-UA node PoC overview

Figure 23 gives an overview about the PoC setup. The central element is the Piccolo TrustNode implementation which is configured as an edge node to integrate a TSN sub-network into the Piccolo ecosystem. The node itself is configured as a docker node which will additionally provide the TSN acceleration capabilities to the Piccolo framework. This enables the deployment of a function which provides an TSN controller with northbound non-realtime interface. The TSN controller itself is used to manage two independent TSN motor drivers to demonstrate the timing accuracy of the system. The components in Figure 23 (from to bottom) fulfill the following tasks:

Piccolo TrustNode: Runs a Piccolo docker edge node implementation with additional TSN hardware acceleration capability.

***REST2OPCUA* container:** This docker container provides the software to be deployed as Piccolo function. It provides the northbound control interface *REST2OPCUA* as described in D2.2 [10]. The interface will be published via the Piccolo agent to be reachable by other functions.

Instance of *REST2OPCUA* function: A running *REST2OPCUA* instance manages a set of TSN devices connected to the southbound interface of the parent node. Therefore it utilises the TSN capabilities of the node to generate time-aware control traffic. In this specific case the control of the TSN devices is realised via OPCUA protocol. The northbound interface *rest2OPCUA* provides the capability to control the TSN devices out of the core network.

RealTimePI: This realtime device receives the OPCUA messages from and generates electrical trigger impulses based on the provided information. Both devices are time synchronized via PTP with the Piccolo edge node. Due to this fact both devices can execute the OPCUA commands exactly synchronously.

Motor drivers with step motor: The trigger signals generated from the RealTimePI are processed by the motor driver to increment the step motor one step forward. Due to the synchronous packet handling of the TSN components both motors will move synchronously.

Oscilloscope : The oscilloscope is used to double check the synchronous operation of both trigger signals generated by the RealTimePIs.

In summary this PoC implementation shows integration of a specialised hardware accelerator into a distributed computing ecosystem. The available resources are managed by the Piccolo agent so that the optimal host for a particular function can be determined. Furthermore it shows the integration of time-sensitive components into a cloud computing environment. Time-critical communication is enabled. By using the OPCUA protocol, also the link between in network computing and industry specific communication is established. The northbound control function can be dynamically used from other in network computing functions.

3.3.3 Development Status

The PoC implementation is currently in the early testing stage. All functional elements, described in Section 3.3.2 are implemented in a minimal functional manner.

3.3.4 Challenges

The main challenge of this PoC is the integration of TSN components into the in network computing ecosystems which are currently not timing aware. A minor challenge is the visualisation of the synchronously rotating step motors of Figure 23.

3.3.5 Next steps

The next steps will be the integration of all components, test and debugging and completing the planned implementations.

4 Conclusion

This report has provided information about the Application Design and Development status of the Piccolo proof-of-concept prototype activities identified in Piccolo Deliverable D1.1. As a result the following use case applications have started to be realized as demonstrations:

- In-vehicle Real-time Behavioural Risk Monitoring
- Smart Factory
- In Network Hardware Acceleration and Time Sensitive Networking

The report has described the work in progress on the journey from concept to working demonstrators. It covers the goals, design, status of the developments, and provides information about the key challenges and next steps to realize those prototypes. These next steps are summarised below for each demonstration.

4.1 In-vehicle Real-time Behavioural Risk Monitoring

Early work on the convergence of the Vision Processing and Vehicular Processing pipelines at the edge of the communication infrastructure has started, with most of the hardware and software components acquired and assembled into a first-form build that is close to "field-ready", by building upon a combination of open source and Commercially-available Off The Shelf (COTS) Bill Of Materials (BOM). For the PoC, we use the Trusted In-Network Computing (TINC) node from Fluentic Networks Ltd. to act as a Piccolo Node (see Piccolo Deliverable D2.2) and to fulfill the security and privacy requirements of both processing pipelines. Open challenges and next steps include:

- Definition of a behavioural risk model derived from the vehicular and vision processing pipelines to support the creation of real-time "Risk Index", for which concepts from the automotive insurance telematics industry are planned to be adopted, in simplified forms.
- The Sensing Feeling Visual Processing Engine (VPE) requires adaptation to operate inside TINC in secure mode due to the complexity of the software build.

4.2 Smart Factory

The target context for the PoC is the "in-network" re-distribution of otherwise centralised Programmable Logic Controller functionality controlling material flow along a complex conveyor belt system in a manufacturing facility. The goal is to demonstrate that distributed controller nodes can be "plugged-in" and automatically cooperate dynamically to self-learn about the conveyor-belt topology and adapt to maximise the throughput of material flow of the whole system. The use case partner has made a facility available, and the selection of components has been made from pre-commercial hardware and software provided by Peer Stritzinger GmbH. Challenges and next steps include:

- Implementation of a simulator and digital twin for function placement and material flow control.
- Design of a distributed plan calculation and execution system.
- Topology learning using link-state like protocols.
- Hardware prototype development based on the GRiSP software stack.

4.3 In Network Hardware Acceleration and Time Sensitive Networking

This PoC activity focuses on demonstrating in-network exploitation of adjunct specialised compute node elements (such as hardware accelerators and GPUs) and Time Sensitive Networking (TSN). The key challenges involved are:

- Specification of additional specialised resources in the Piccolo API
- Incorporation of TSN features into the Piccolo node as described in D2.2.

References

[1] Piccolo Project. *Use cases, Application Designs and Technical Requirements*. Deliverable D1.1. 2021.

[2] *Eclipse Kuksa*. https://www.eclipse.org/kuksa/. Accessed: 2021-09-21.

[3] *Genivi IoT Event Analytics*. https://github.com/GENIVI/iot-event-analytics. Accessed: 2021-09-21.

[4] GENIVI Alliance. *Genivi/W3C Vehicle Signal Specification*. https://genivi.github.io/vehicle_signal_specification/. Accessed: 2021-09-21.

[5] Sensoris. *Sensor Interface Specification*. https://sensoris.org/. Accessed: 2021-09-21.

[6] International Electrotechnical Commission. *IEC 61131 Programmable Controllers - Parts 1-10*. 2003.

[7] International Electrotechnical Commission. *IEC 61499 Function blocks - Parts 1-4*. 2012.

[8] Piccolo Project. *Piccolo Node Definition*. Deliverable D2.1. 2021.

[9] Amin Abdul. *Hierarchical CUC/CNC management model*. https://www.ieee802.org/1/files/public/docs2020/60 abdul-hierarchical-CUC-CNC-management-model-0520-v02.pdf. May 2020.

[10] Piccolo Project. *Initial Report on PoC Implementation of a Piccolo Node*. Deliverable D2.2. 2021.