Performance Engineering of Cyber-physical Systems in the Compute Continuum

RT-Cloud 2023 Keynote

Prof. dr. Benny Akesson

ESI

Powered by industry, academia and TNO

Examples of Dutch Cyber-physical Systems (CPS)

System Complexity is Increasing!

Five technological and market trends drive increasing complexity in CPS:

1. **Additional functionality**
   - Number of interfaces and lines of code are **rapidly increasing**
2. **Mass customization**
   - Increased customization of systems at design time to the point where **each system is unique**
3. **Long life times**
   - Systems operate for decades and need to **continuously evolve** after deployment
4. **Increasing autonomy**
   - Systems acting autonomously with **little or no human interaction**
5. **Systems of systems**
   - Interconnected systems of which **nobody is in complete control**

Managing Complexity

Increasing complexity cannot be dealt with by current engineering methodologies

- Increasing **development and maintenance costs**
- Increasingly hard to **guarantee functional correctness** and **balance system qualities**
- Severe shortage of **skilled people**

New design methodologies are required to manage the increasing complexity and enable future generations of CPS to be developed efficiently!

TNO-ESI at a Glance

ESI
Powered by industry, academia and TNO

SYNOPSIS

- Foundation ESI started in 2002
- ESI acquired by TNO per January 2013
- ~60 staff members many with extensive industrial experience
- 8 Part-time professors

FOCUS

Managing complexity of high-tech systems

through

- system architecting
- system reasoning and
- model-driven engineering

delivering

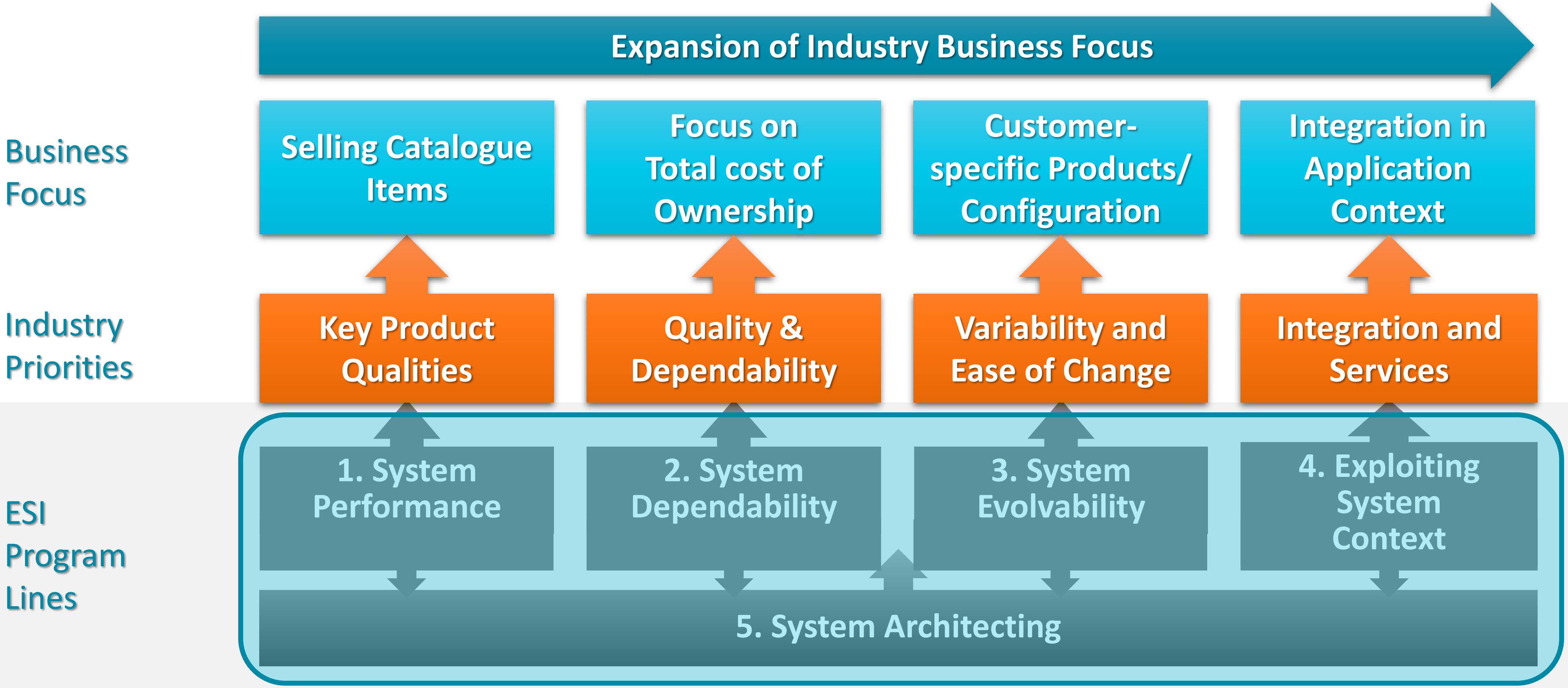
- methodologies validated in cutting-edge industrial practice

PARTNER BOARD

ASML
Canon
CANON PRODUCTION PRINTING
itec
PHILIPS
Radboud University Nijmegen
THALES
ThermoFisher SCIENTIFIC
TNO
TU Delft
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF TWENTE.
UNIVERSITY OF AMSTERDAM
VANDERLANDE

Capgemini engineering

From Industry Focus to Program Lines



System Performance

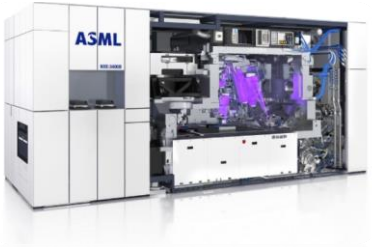
ESI has many years of experience in the area of system performance

- System performance is the amount of work done by a system at a predefined quality level
- Considers performance in terms of **timing**, e.g. latency and throughput

Work considers the complex cyber-physical systems of our partners

- Typically distributed systems with a **monolithic component-based software architecture**

… but increasing complexity is driving change …






Different Performance Characteristics in the ESI Eco-system

ESI
Powered by industry, academia and TNO

The world we know

	ASML	Thales	Philips
Distribution scope	Device	Device	**Device-edge-cloud**
Software architecture	Monolithic component-based	**Microservices**	**Mixed**
Problem	Fine-grained performance analysis/diagnostics	Traceability of (performance) requirements / Performance verification	Performance analysis, service continuity
Order of timing requirements	Micro-/milliseconds	Hundreds of milliseconds	Seconds
Timing requirements	Firm	Firm	Soft

The world of some of our partners

The future of our partners?

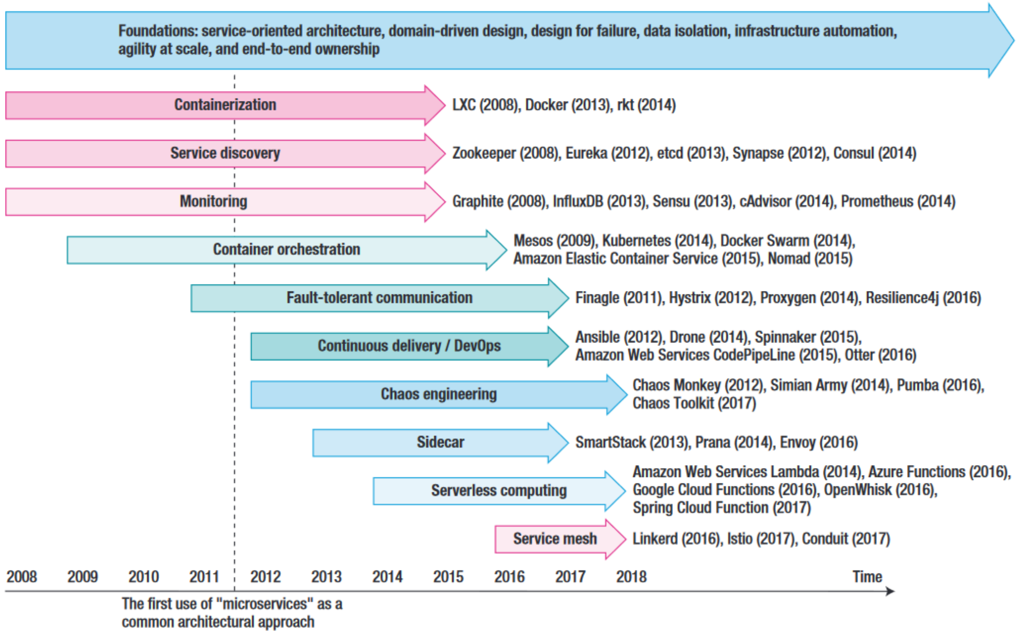
Problem Statement

These changes challenge our expertise

- Do our methods and tools for performance engineering translate?

We have limited experience with:

- **microservice technologies**
  - E.g. containers and their orchestration, sidecars and service meshes
- **public/private cloud environments**
  - E.g. pod/node scaling



[1] Jamshidi, Pooyan, et al. "Microservices: The journey so far and challenges ahead." *IEEE Software* 35.3 (2018): 24-35.

Technology in this area is developing rapidly!

Presentation Outline

Powered by industry, academia and TNO

- Introduction
- Performance Verification in Microservice Architectures
- Performance Analysis and Service Continuity of CPS in the Compute Continuum
- Conclusions

Performance Verification in Microservice Architectures

Thales Case Study

Running Dutch PPP Project

System Context

Thales is making a new product to re-establish itself in the fire control market

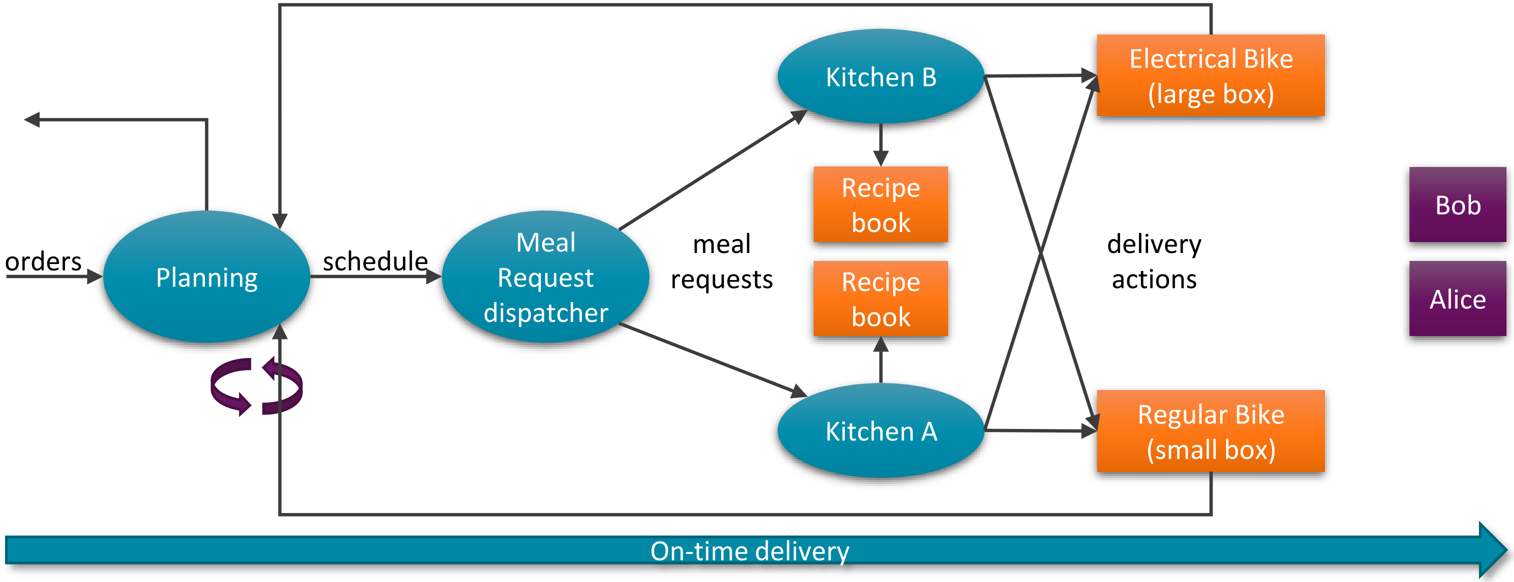
The platform is a microservice architecture running in a private cloud environment

- Software is decomposed into **independent services** communicating through **message passing**
- Makes use of **containerization** to improve portability and increase deployment options
- Uses **container orchestration** to improve system resilience

Research Context

The considered application for this work is a "Thales Meal Delivery System"

- Inspired by a real application, but abstracted to **protect sensitive IP** and allow **public dissemination**



Performance Verification Problem

A gap between the disciplines of system and software engineering has been observed

- Makes it hard to trace whether requirements are satisfied during design, operation, and evolution

Systems are specified at logical level, but are implemented and verified at the physical level

- The **translation** between these two levels is **manual** and requires **substantial effort**, and must be repeated if either level **evolves** during development or operation
- Timing requirements of system flows are **specified early**, but typically only **verified in late stages of development** when changes are more **time consuming** and **expensive** to make
- It is difficult to verify that the software implementation **conforms** to the system specification

Example System Flow – Timing Requirement 250 ms

ESI
Powered by industry, academia and TNO

Automated Telemetry-based Performance Engineering

We envision an automated approach to telemetry-based performance engineering

- Enables performance to be addressed **frequently** and **consistently** during the system life-cycle
- Nightly **performance verification** to determine whether timing requirements are satisfied
- Nightly **conformance checking** that detect whether implementation and specification diverge

To implement the vision, we need three key ingredients:

1. Timing requirements must be **formally specified** at logical level
2. Interactions between services in the system to be **observable**, e.g. through tracing
3. Relevant interactions in traces must be **identified** and **extracted** to verify performance requirements and check conformance

Considerations for Methodology

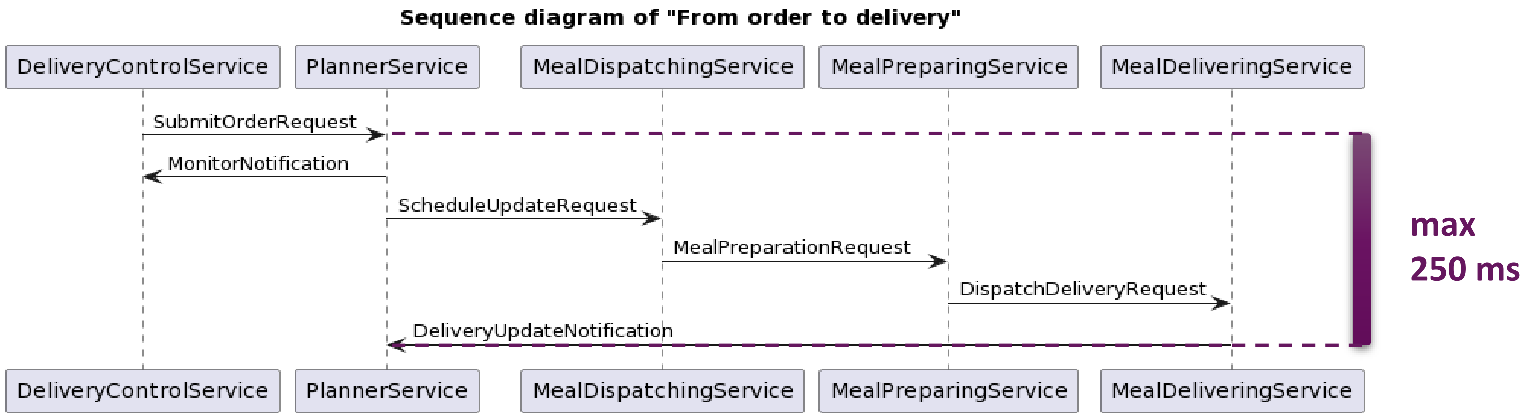
Five considerations for verification methodology:

1. Enable gradual introduction of verification methodology to simply adoption
2. Use already known or available specifications to simplify adoption
3. Use a formal specification to prevent ambiguity
4. A clear relation between specification and observable data is required for traceability
5. Use a model-based approach to enable automation

PlantUML Specification of Timing Requirements

Sequence diagrams were chosen as the specification language

- **Well-known** and **commonly used** formalism among system and software engineers at Thales
- **Common format** (PlantUML) and **open tooling** can be used
- Timing requirements added in PlantUML comments to **avoid changing grammars**




```
@startuml
title
Sequence diagram of "From order to delivery"
end title

participant DeliveryControlService
participant PlannerService
participant MealDispatchingService
participant MealPreparingService
participant MealDeliveringService

'@TimingStart 250000
DeliveryControlService-> PlannerService:
SubmitOrderRequest
PlannerService -> DeliveryControlService :
MonitorNotification
PlannerService -> MealDispatchingService :
ScheduleUpdateRequest
MealDispatchingService -> MealPreparingService :
MealPreparationRequest
MealPreparingService -> MealDeliveringService :
DispatchDeliveryRequest
MealDeliveringService -> PlannerService :
DeliveryUpdateNotification
'@TimingEnd
@enduml
```

ESI

Powered by industry, academia and TNO

The Pillars of Observability

Observability is provided through three complementary types of telemetry data

1. **Metrics provide time-based numerical measurements on elements of the application or system**
   - **System metrics:** CPU/memory/disk/network utilization, network retries, # deployed pods
   - **Application metrics:** # of exposed wafers, # of failed payments, customer satisfaction

2. **Logging collects application-generated structured or unstructured text**

3. **Traces represent flows through the system in services and function calls**

[1] Goniwada, Shivakumar R. "Observability." *Cloud Native Architecture and Design*. Apress, Berkeley, CA, 2022. 661-676.
[2] Li, Bowen, et al. "Enjoy your observability: an industrial survey of microservice tracing and analysis." *Empirical Software Engineering* 27.1 (2022): 1-28.

Proposed Observability Solution

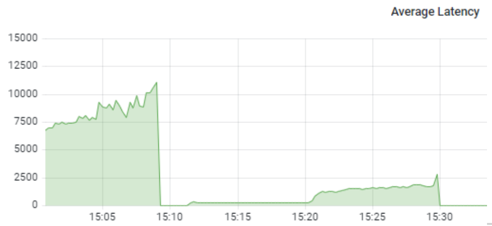
We have proposed an observability solution based on:

- **Prometheus** for **monitoring** and **alerts** for application and system **metrics**
- **Jaeger** for **distributed tracing**
- **Grafana** for **visualization** across data sources
- **OpenTelemetry** for **technology-agnostic instrumentation**

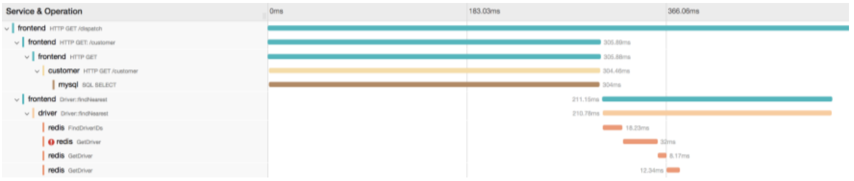
This direction is chosen because:

- Thales software architecture uses **containerized software** in a **Kubernetes environment**
- Thales has a preference for **open source tools** maintained by large communities

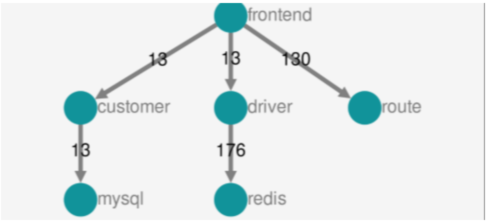
Metric dashboards



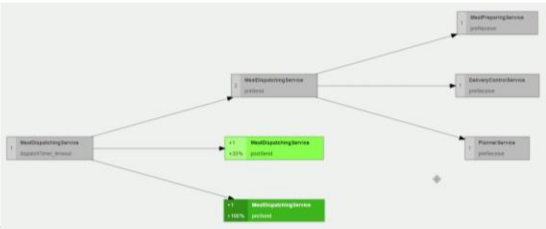
Visualization of nested spans



Service dependency graphs



Trace comparison



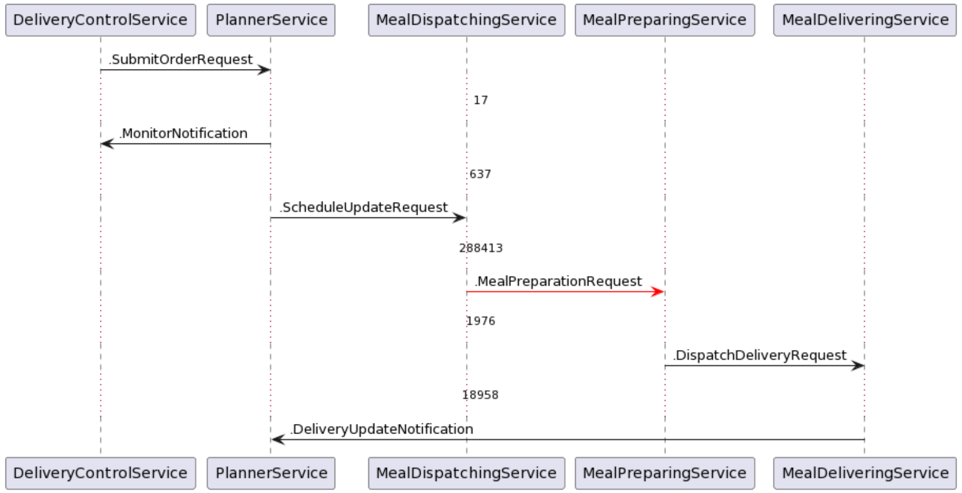
Performance Verification

Relevant interactions in traces are identified based on the sequence diagrams

- Conversion by using PLY, a Python based library for compiler construction
- PLY uses the PlantUML BNF grammars to create an abstract syntax tree (AST)
- Transformation from AST creates a list of service interactions to be extracted from traces

Interactions are extracted and verified

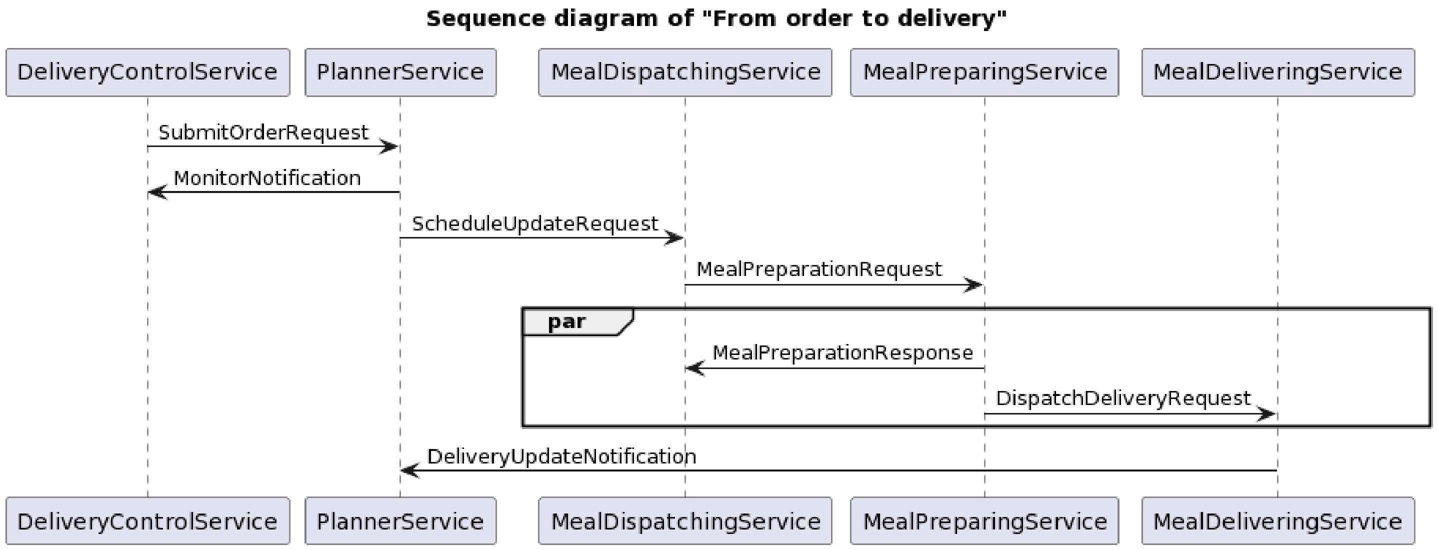
- Listed service interactions are extracted from ElasticSearch and requirements are verified
- If a requirement is violated, a PlantUML diagram is generated with relevant timing information
- More refined user feedback is being discussed



Next Steps

Currently, this work is being extended to support conformance checking

- Extension of the same infrastructure
- **PAR statements** in PlantUML used to indicate that some interactions can happen in arbitrary order
- We will find out if the simple PlantUML specification scales to cover this case in a good way



Performance Analysis and Service Continuity of CPS in the Compute Continuum

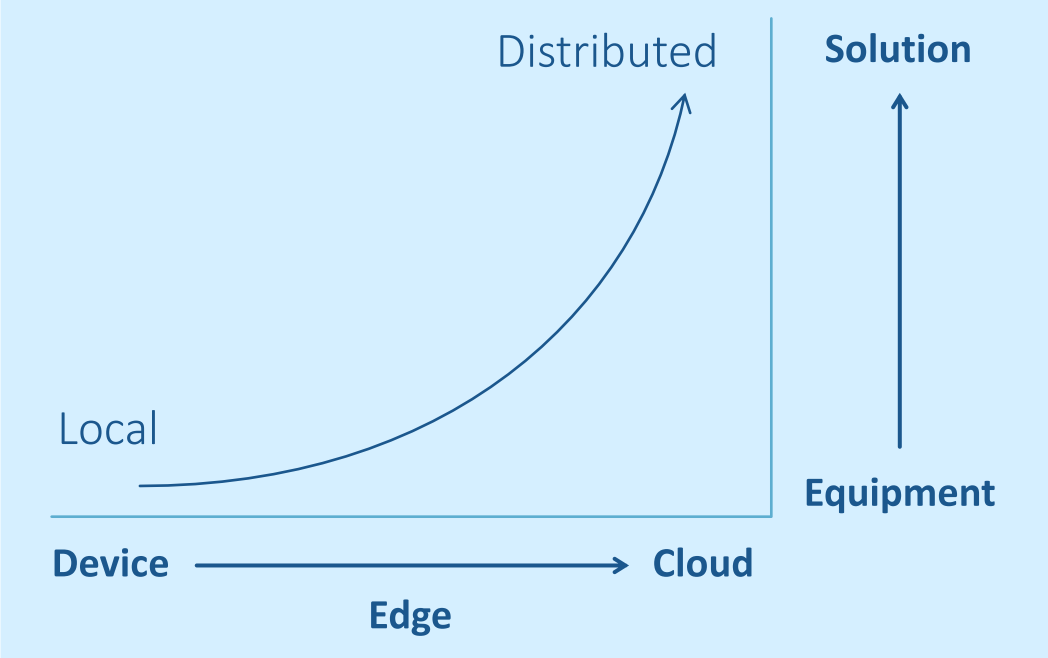
Philips Case Study

Running ECSEL Joint Undertaking

TRANSACT Project Goal

Develop a universally applicable distributed solution architecture, framework and transition methodology for the transformation of standalone safety-critical CPS into distributed safety-critical CPS solutions.

1. Transforming CPS’ architecture from monoliths to distributed solutions
2. **Ensuring CPS’ performance and safety in the device-edge-cloud continuum**
3. Ensuring CPS’ security and privacy in the device-edge-cloud continuum
4. Devising business models for CPS deployed in the device-edge-cloud continuum



[1] Hendriks, Teun, et al. "Thirteen concepts to play it safe with the cloud." 2023 IEEE International Systems Conference (SysCon). IEEE, 2023.

TRANSACT Use Cases

Remote operation of autonomous vehicles for navigating in urban environments

- Reduce road fatalities and accidents
- Contribute to a more efficient urban mobility with less congestion
- Reduce fuel cost and GHS-emissions

Critical maritime decision support enhanced by distributed AI-enhanced edge and cloud solutions

- Reduce groundings and other incidents
- Increase performance
- Reduce fuel cost and GHS-emissions

TRANSACT Use Cases

Cloud-featured battery management for electric vehicles

- Increase electrification of car park
- Reduce air pollution

Critical wastewater treatment decision support enhanced by distributed, AI-enhanced edge and cloud solutions

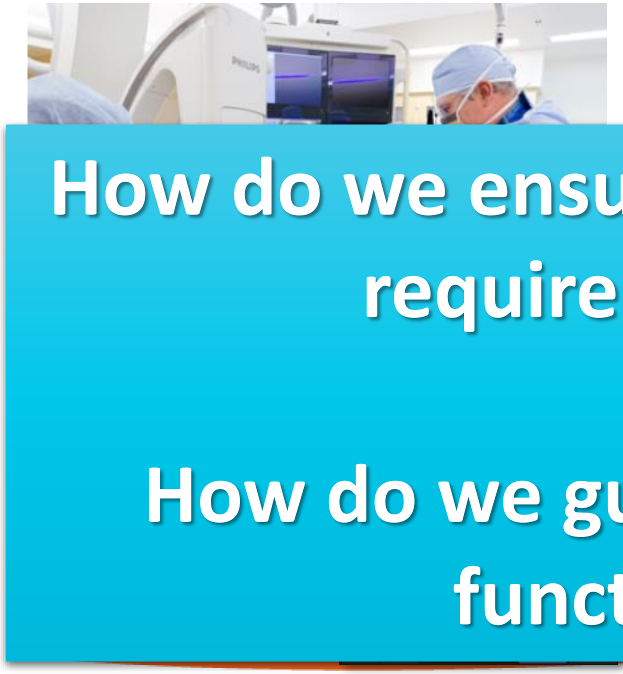
- Mitigate climate change induced water scarcity
- Prevent ecological disasters due to potential wastewater spills

Edge-cloud-based clinical application platform for image-guided therapy and diagnostic imaging systems

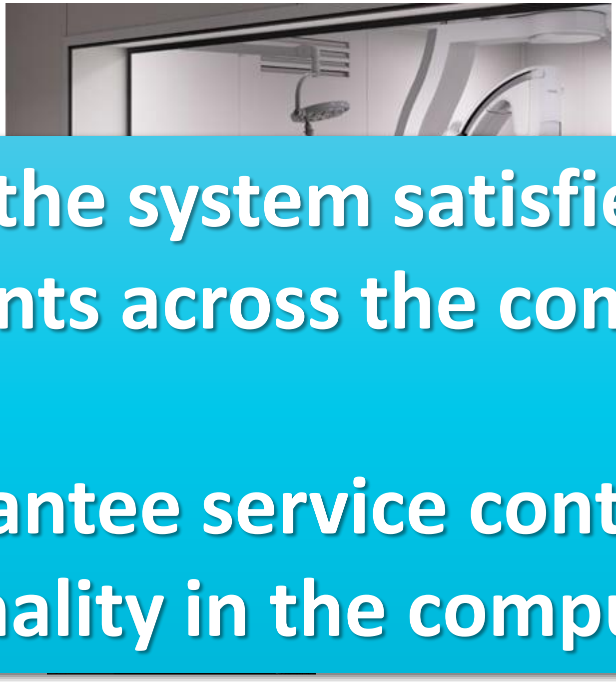
- Better clinical outcomes at lower cost
- Increased medical staff’s experience
- New business models based on 3rd party tool integration

Philips (IGT/HSDP) Case Study

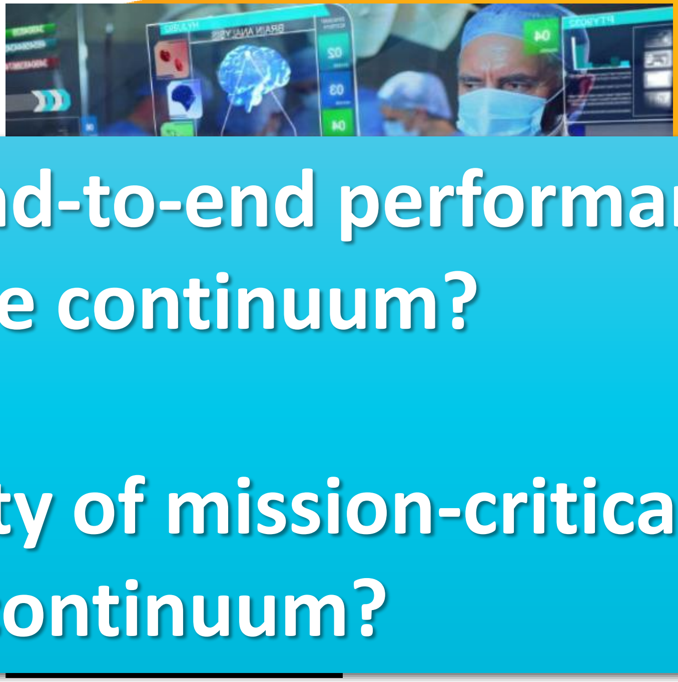
Device tier



Edge tier



Cloud tier



How do we ensure the system satisfies end-to-end performance requirements across the compute continuum?

How do we guarantee service continuity of mission-critical functionality in the compute continuum?

Safety-critical, real-time image & data acquisition and viewing

Pre-interventional planning or off-line reviewing, cloud enhanced intervention applications

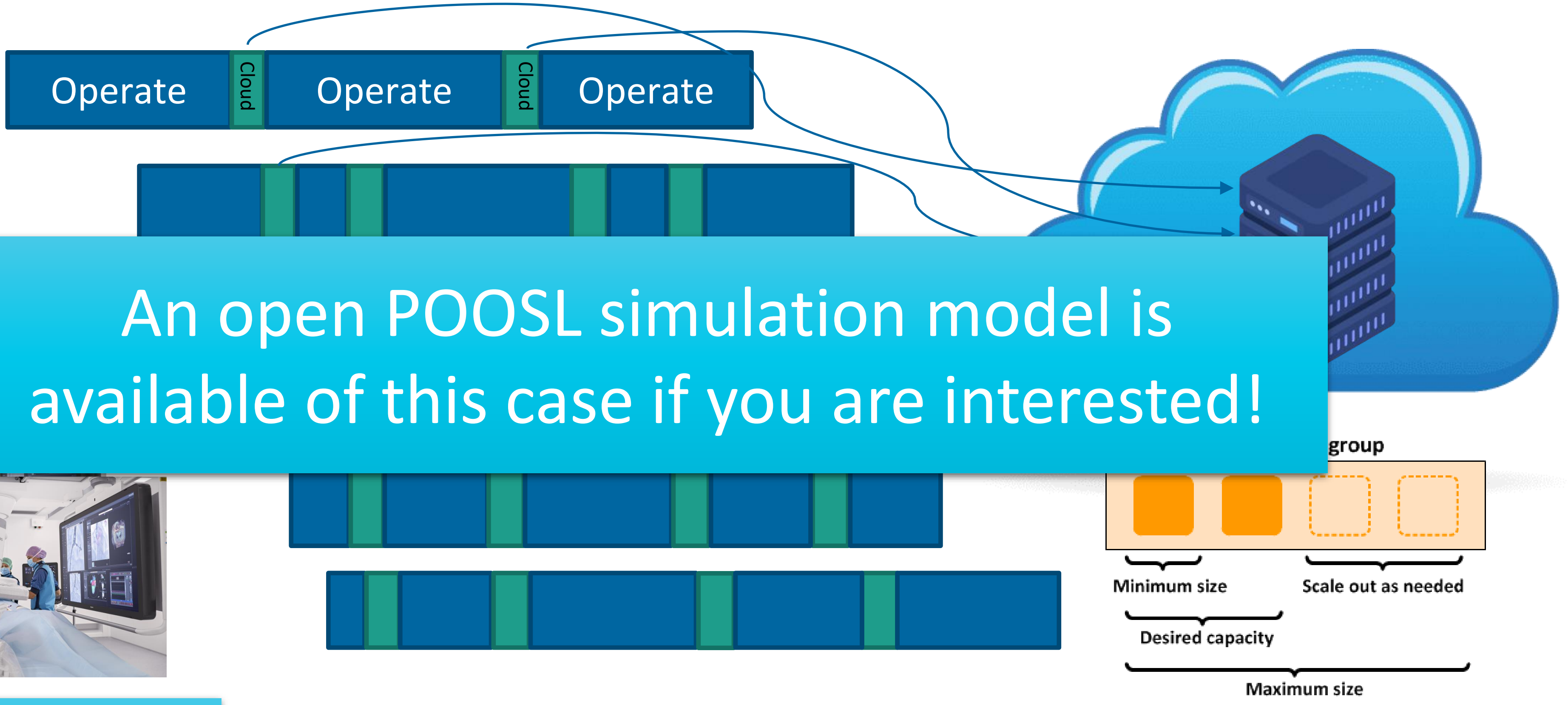
Data access enabler for AI applications, predictive analytics, download upgrades, 3rd party integration and enhanced services

Cloud-assisted Image Guided Surgery

Advanced image processing in the cloud, while patient is on the table
Response time should not keep surgeon waiting

How to meet SLA for Response Time?

Impact of dynamic demands on auto-scaling needs for cloud resources



An open POOSL simulation model is available of this case if you are interested!

So many hospitals?
So many operations?
So many cloud requests?

Stochastic, time-varying demand

How many cloud resources?
How to scale to meet timing?

TRANSACT Open Experimental Platform

ESI
Powered by industry, academia and TNO

Open experimental platform for applied research and performance benchmarking

Platform technologies:

- AWS for public cloud environment
- TNO private cloud environment with 3 nodes
- Kubernetes for container orchestration
- Prometheus for monitoring of application and system metrics
- Jaeger for distributed tracing
- Grafana for visualization and alerts across data sources
- OpenTelemetry for technology-agnostic instrumentation

Open Source 3D Reconstruction Application

A folder of 2D images are selected on a client application running in a web browser

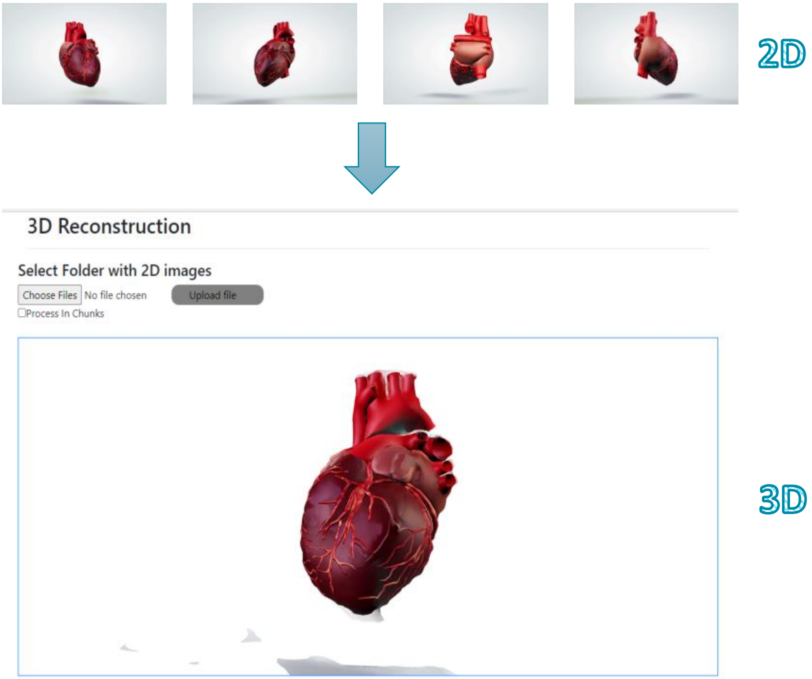
- The images in selected folder are uploaded to an S3 bucket

OpenMVG (Open Multiple View Geometry) matches the images and creates a sparse point cloud

- Wrapper allows batches of images to be processed in parallel

OpenMVS (Open Multi-view Stereo) performs 3D reconstruction based on the point cloud

The reconstructed 3D model is shown in the web browser of the client application



Features of 3D Reconstruction Demonstrator

Instrumentation and integration with observability tools

- Instrumented with **OpenTelemetry** to get **spans, traces and latency data**
- Integration with **Prometheus** and **Grafana** for **metrics, visualization, and alerting**

Parallel processing of images to benefit from autoscaling

- **Horizontal pod autoscaling** using **Kubernetes**
- **Horizontal and vertical node autoscaling** using **Karpenter**
- Scaling is **too slow** to make a difference for a single job, but it useful at fleet level

Service continuity in case of lost network connection through mode switch to device-only mode

- Device and cloud clusters are **securely linked** using **Skupper**
- **Load balancing** and **failover** is handled by **Nginx**
- Switching from cloud mode to device-only mode, and vice versa, takes approximately **10 seconds**

Next Steps

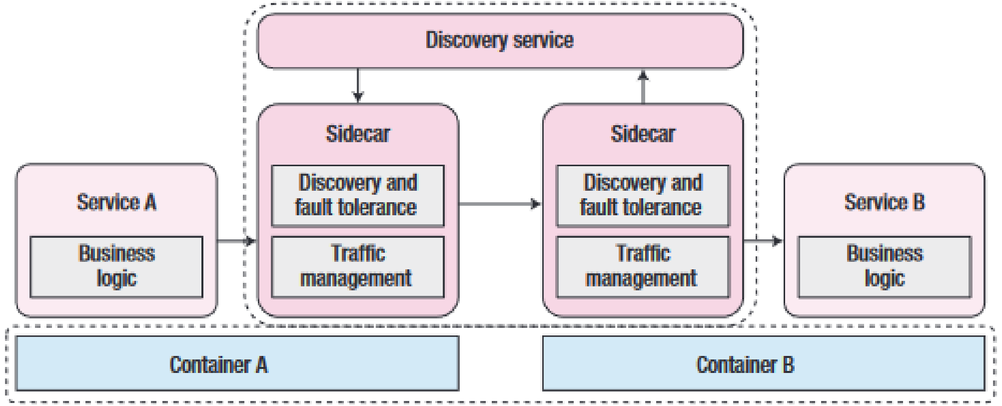
Quantify, measure and control performance overhead and quality of observability

Comparison of presented technology to a service mesh implementation [1, 2]

- Services communicate via sidecar proxies, forming a data plane, configured via a control plane

Fundamental features of a service mesh:

1. Service discovery
2. Load balancing
3. Fault tolerance
4. Traffic monitoring
5. Circuit breaking

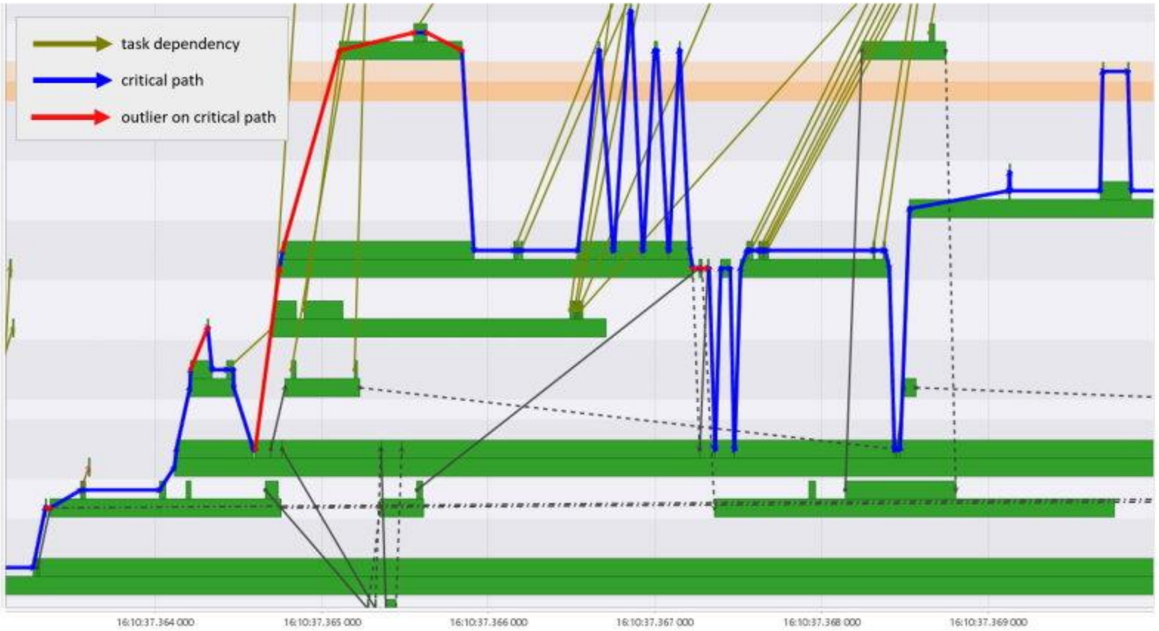


[1] Jamshidi, Pooyan, et al. "Microservices: The journey so far and challenges ahead." *IEEE Software* 35.3 (2018): 24-35.
[2] Li, Wubin, et al. "Service mesh: Challenges, state of the art, and future research opportunities." 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE, 2019.

Next Steps

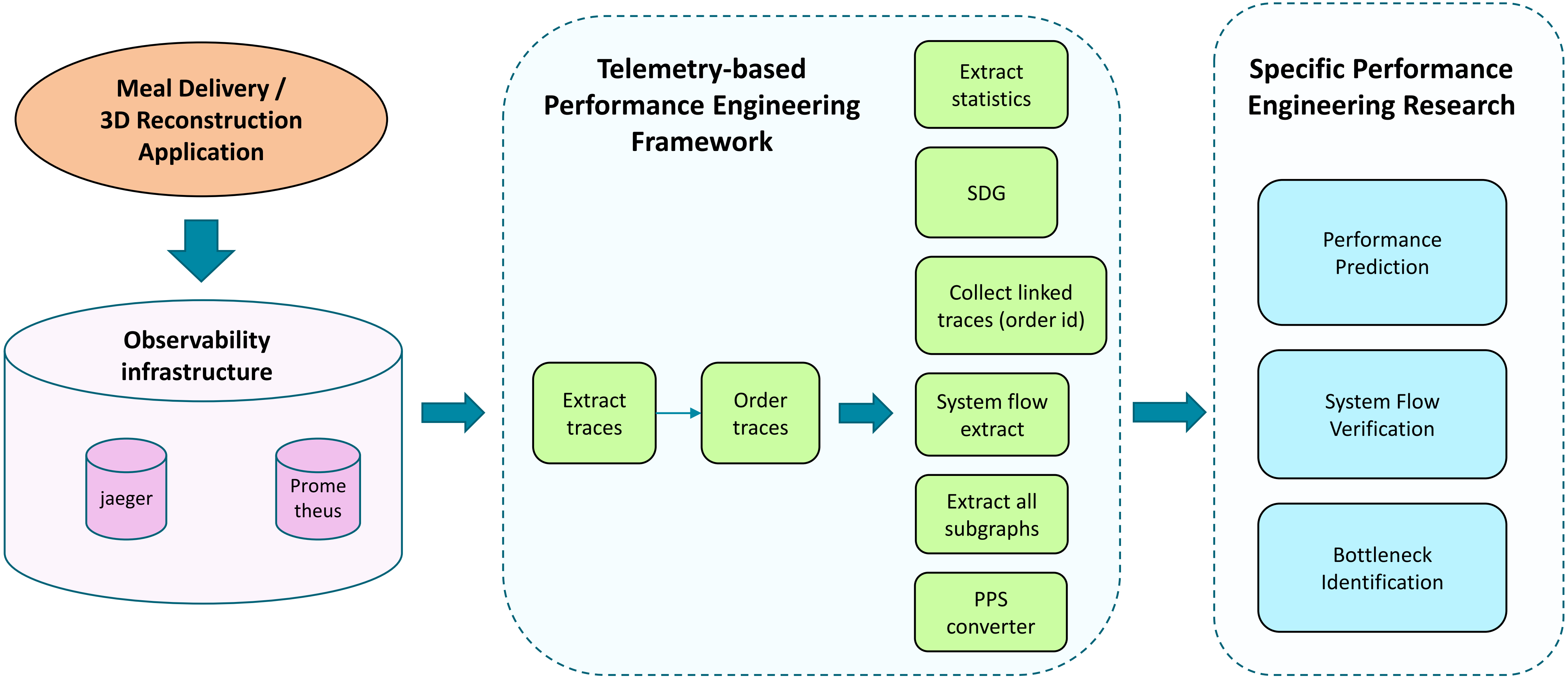
Observability alone is not sufficient in a system with many concurrently executing services [1]

- Automatic **root-cause analysis** and **critical path analysis** leveraging formalization between traces and **timed-message sequence charts** (TU/e) and the **Platform Performance Suite** (PPS) by ESI [2]



[1] Bento, Andre, et al. "Automated analysis of distributed tracing: Challenges and research directions." Journal of Grid Computing 19 (2021): 1-15.
[2] Bits & Chips, “Clearning the critical software path”,(2021) https://bits-chips.nl/artikel/clearing-the-critical-software-path/

Overview of Performance Engineering Framework



Conclusions

Conclusions

The complexity of cyber-physical systems is increasing

- Driven by increasing needs for **functionality**, **customization**, **evolvability**, and **autonomy**, as well as integration in a broader **system-of-systems** context

New model-based design methodologies are needed to enable the next-generation of CPS to be efficiently developed, reducing development time and improving system quality

The architecture of (some) cyber-physical systems is changing

- From component-based to **service-oriented/microservice** architectures
- From distributed within a device to distributed over the **cloud continuum**
- This challenges **ESIs experience and expertise** in performance engineering

Conclusions

We discussed two relevant case studies from running projects:

1. **Performance Verification in Microservice Architectures**
   - **Thales Meal Delivery system** on a **microservice architecture** in a **private cloud environment**
   - A methodology based on **open source tools** allows requirements specified using **sequence diagrams** to be verified using **telemetry data**

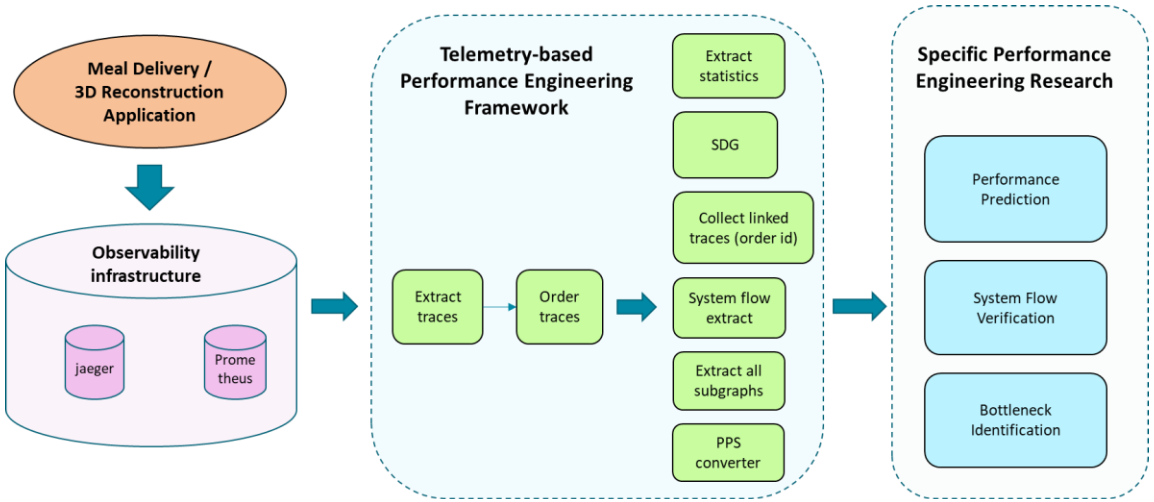
2. **Performance Analysis and Service Continuity of CPS in the Compute Continuum**
   - **Philips x-ray solution** distributed over **compute continuum**
   - Discussed alternatives for providing **end-to-end performance** and **service continuity** in continuum

Conclusions

There are clearly good open source tools to manage performance and service continuity in microservice / cloud applications

- The challenge is to provide **meaningful guarantees** on their timing behavior

From this work, a general telemetry-based performance engineering framework is emerging



Acknowledgements

This work was partially supported by the TRANSACT EU project, ((https://transact-ecsel.eu/). TRANSACT has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Belgium, Denmark, Finland, Germany, Poland, Netherlands, Norway, and Spain.

The research is carried out as part of the ArchViews project under the responsibility of TNO-ESI with Thales Nederland B.V. as the carrying industrial partner. The ArchViews research is supported by the Netherlands Organisation for Applied Scientific Research TNO.