The MANTIS Architecture for Proactive Maintenance

Csaba Hegedűs
AITIA International Inc.
Telecommunication Division
48-50 Czetz János str., Budapest, Hungary, H-1039
Email: hegeduscs@aitia.ai

Pál Varga, István Moldován
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics
2 Magyar Tudósok krt., Budapest, Hungary, H-1117
Email: {pvarga,moldovan}@tmit.bme.hu

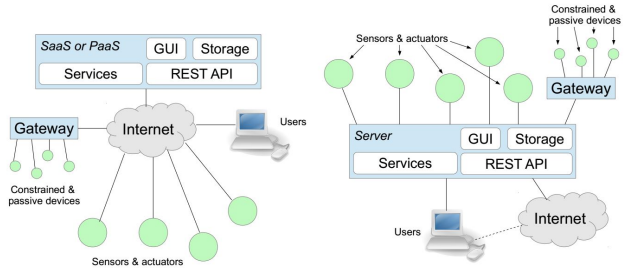
***Abstract*—Data collection, processing and visualization techniques has been going through a rapid evolution in recent years. Various applications utilize these new results; especially combined. Parallel to this, there are new developments within the Cyber-Physical Systems (CPS) domain. Beside the main purpose of CPS – getting physical systems serve to work better, faster, more optimized –, the concept can improve the long-term usability of the physical equipment, as well. The principles of proactive maintenance are rooted from the need of short-term fault correction and long-term usability. The idea is to track system status not only for operations but for maintenance purpose as well. This allows for scheduling maintenance based on need – rather than based on operating time or "milage". This paper presents the MANTIS framework for proactive maintenance. It utilizes CPS concepts for system modeling, furthermore, it proposes a combined toolset for data collection, processing and presentation. In order to cover the full value chain, the framework applies for all the three Tiers: the Edge, the Platform, and the Enterprise, as well.**

I. Introduction

The purpose of this work is to describe the MANTIS reference architecture. The main objective of the ECSEL MANTIS project [1] is to develop a Cyber Physical System (CPS) based Proactive Maintenance Service Platform Architecture enabling Collaborative Maintenance Ecosystems. The generic focus is on an architecture that enables service-based business models and improved asset availability at lower costs through continuous process and equipment monitoring, together with data analysis. This architecture takes into account needs of various industries in the forefront of service-based business and operations – as well as less mature ones so improvements in maintenance can be achieved gradually and consistently. The higher level requirements for the whole project are described by [1] and further tuned for the architecture in [2].

CPSs are nowadays built together within some form of Internet of Things (IoT) architectures. A "usual" IoT application scenario involves various *things* collecting enormous amounts of data from various places and sending them to a *cloud* for a specific purpose. A survey of 39 IoT platforms [3] concluded in a generic architecture and common characteristics of IoT platforms. Figure 1 depicts a generalized commercial IoT platform in its fullest form: in here, two possibilities are shown. The various IoT modules and services can be deployed on local premises – or within a global IT cloud.

The generic modules in such a platform are the following:



(a) Cloud-based platform (b) Local platform

Fig. 1. A generalized IoT framework [3]

- Sensor or actuator nodes, motes that create and then send in the data – or act based on the received data.
- Gateways that "hide" constrained devices that might be communicating via non-IP based networking ("legacy") and/or incapable of implementing the platform interface.
- A platform interface that receives the data from the devices and passes it to other modules (gateway and data distributor).
- Data storage, often distributed, which is accessible by other modules of the platform.
- Various service modules that can access the historical or even the current inbound data streams and generate insights and various processing tasks (i.e. "big data applications").
- Graphical interfaces for operators to manage the system and validate the output (i.e. "Business Insights").

The current paper is organized as follows. First, it briefly describes the requirements for the MANTIS architecture addressing CPS-based proactive maintenance. After this, it refreshes some related work and technologies – in order to effectively describe the data collection, processing and presentation architecture. The modues of the architecture fall into the generic categories of the above listing – which is not a coincidence, but a logical reflection of the essential best practices gathered within the domain.

II. Scope and Requirements for the MANTIS Architecture

The MANTIS architecture is expected to enable proactive maintenance: estimate future performance, predict and prevent imminent failures and even schedule maintenance proactively. One of the biggest encountered issues lies with the nature of legacy data collection processes. Mostly, the historical data in the various use cases is not suitable for the purposes of proactive maintenance, often lacking annotation, have missing values or simply badly structured. Hence, this intended use is an industrial application of IoT fundamentals and concepts: the major motivator for such a system is the possibility for rapid development of non-invasive monitoring of various assets.

A reference architecture provides a template, often based on the generalization of a set of solutions. This is also the case here for MANTIS. These solutions are often generalized and structured for presenting one or more architecture structures based on the harvesting of a set of patterns that were observed in a number of successful implementations. Similarly, the MANTIS architecture has been created by analyzing commonalities and variabilities of the specific use case implementations within the project, abstracting realization details and adding solutions for missing key functions and quality aspects.

A. Proactive maintenance target areas within MANTIS

The purpose of the MANTIS architecture is to make proactive maintenance possible in a scalable, multi-leveled way. We are targeting condition-based maintenance: the processes are defined by the ISO 13374 standard [4]. Since large amounts of data can be collected from industrial devices, machines or vehicles, there is sense in trying to utilize them using emergent technologies. In order to enable maintenance optimization and new business models with MANTIS, appropriate utility services are developed. Within these, data mining and analytic services are created, which are mainly related to these functions [2]:

- Remaining Useful Life (RUL) of components: continous tracking of telemetry (usage) data and estimating how much time the given device or component has left before needs to be replaced,
- Fault Prediction (FP): the system shall predict based on diagnostic data an inbound failure mode (different to wear-out to be detected by RUL),
- Root Cause Analysis (RCA): when an unpredicted, complex failure occurs, the system shall deduct the actual module, the rout caused of the issue, and
- Maintenance Strategy Optimization (MSO): provide a decision making support on better maintenance planning.

B. Specific requirements related to the architecture of MANTIS

Due to the different industrial domains addressed, there were altogether over 530 different requirement-items that got collected from various project partners. This is a striking number by itself, and there is no trivial solution on how to tackle each individual requirement. First, these had to be categorized.

Many of the requirement categories were related to the *operating environment* of the MANTIS architecture: communication restrictions and expectations, design principles, the need for web clients, integration of legacy human-machine interfaces, and so on. These have not been directly addressed by MANTIS, although, when designing the architecture, we had to keep in mind that the final, integrated systems has to cover these as well. All other requirements (over 150 high-level items) that fell into the MANTIS focus have been addressed one-by one. These categories included (i) data handling, (ii) event handling, (iii) guarantee-related, and (iv) security issues.

The most important design requirements are aimed towards scalability, fault tolerance in the data collection and processing. The inputs of the platform are coming from so-called edge devices,and the output is utilized by various enterprise systems or personnel. It is worth noting that the scope of MANTIS platform architecture does not include or target the actual lifecycle management of the devices. To do that, MANTIS relies on the already existing corporate systems, resources. However, these interactions with external systems is planned and designed into the framework via standardized secure communications between platform modules.

III. Related Work and Technologies

A. Reference Architecture for the Industrial Internet of Things

The data gathering and processing viewpoit shown by Figure 1 corresponds to that of the IIoT reference architecture proposed by IEC [5]. However, three additional architecture patterns are utilized to better suite the targeted environments, extending the general standalone IoT solutions. These include (i) a three-tier architecture pattern, aided by (ii) gateway-mediated edge processing, and a (iii) layered databus pattern. This latter term is related to one of the biggest value added of the IIoT approach: reusing "legacy" production systems, by "making them smarter" with additional, usually non-invasive components (i.e., connecting to their management interfaces).

Nevertheless, this requires that physical machines are to be connected to the network, their operations offered as "services" – hence creating cyber-physical systems (CPSs [6]) out of them. This requirement invokes the need for data buses, that create a logical space that implements a set of common schema and communicates using those set of schema between endpoints (i.e. translation between various data description ontologies). Therefore a data bus supports communication between applications and devices: semantics and translation are the basis for interoperability within IIoT.

Moreover, this architecture is therefore dissected into three Tiers. The first one is the Edge Tier, where the sensors and actuators are located (e.g., production floors), and might tap into the communications within the (real-time) control loops between the given CPSs (cf. ISA95 systems [7]). In here, we are collecting mostly process telemetry, then aggregating and preprocessing it locally. This is supported by an application gateway that provides connectivity: it bridges to a wide area network towards the platform level. It also acts as an endpoint for the wide area network while isolating the local network of edge nodes (i.e., the involved local CPSs). This architecture pattern allows for localizing operations and controls (i.e., edge analytics and computing). Its main benefit is in breaking down

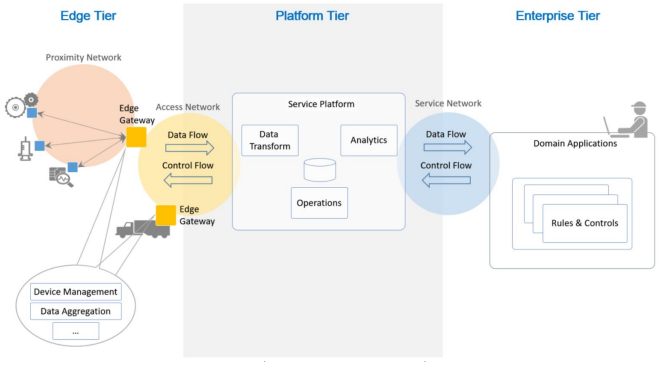


Fig. 2. The Industrial Internet of Things reference architecture

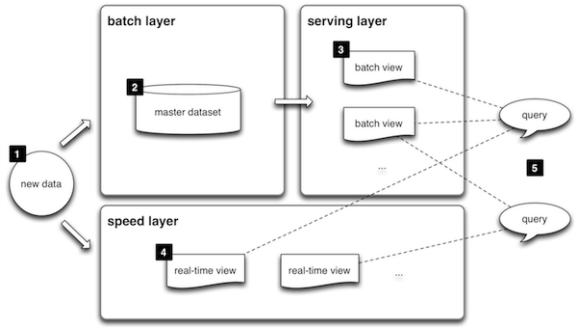


Fig. 3. The Lambda data processing architecture

the complexity of IIoT systems, so that they may scale up both in numbers of managed assets as well as in networking. The Platform Tier receives, processes and forwards control commands from the Enterprise Tier to the Edge Tier. It consolidates and analyzes the data flows from the Edge Tier and other systems. It provides management functions for devices and assets. It also offers non-domain specific services such as data query and analytics. The functional blocks of the cloud platform are the same as in generic IoT platforms. Meanwhile, the Enterprise Tier receives the processed data flows (i.e. business insights) from the Edge and Platform Tiers. It might also issue control commands to the Platform and Edge Tiers.

B. Data Processing in Lambda

The primary purpose of any (I)IoT systems is to create value added by processing the collected data in a cloud platform. According to the generalized Lambda architecture pattern [8] defined by industry experts, data can be processed either as soon as it reaches the platform (stream processing), or later on, on demand fetched from storage (batch processing). Figure 3 depicts the overview of a generic analytic platform. In here, the "speed layer" comprises of stream processing technologies, that are processing inbound data real time. This is an event-driven programming paradigm, where the processing functionality receives touples periodically, and executes the same function over them (e.g., creating a counter for a specific message type or classifying them using a well-taught machine learning algorithm). The other type of processing is asynchronous to the inbound data, and can be called on batched, already stored datasets. These tasks are run once at a time and might take long to complete – such as the training phase of a machine learning algorithm. An other major responsibility of the batch layer is to maintain the (distributed) data storage, aided by the serving (database) layer. These modules are naturally part of any IIoT application. Many commercial products and platforms support these operations.

C. Maintenance Based on MIMOSA

Within MANTIS, the MIMOSA [9] is providing the common understanding and data ontology between partners and applications. MIMOSA was developed in an ISO-13374 Standard (Condition Monitoring and Diagnostic of Machines) compliant manner, being an implementation of the latter's functional specifications. It is presented as a defining standard format for the data exchange, while it also provides the data meta-model structure together with the definition of the ontologies of the data. In fact, one of the greatest benefits in using MIMOSA is this definition of semantics and ontologies so that parties developing their MANTIS solutions do not need to worry about how different types of information need to get linked together. In here, therefore, MIMOSA serves the role of the common databus [5], while remaining loosely coupled. MANTIS has developed a full stack of message models extending the MIMOSA ontology that are designed to facilitate communications between edge and cloud, and also between the various cloud modules.

Moreover, MANTIS follows the ISO-13374 standard in terms of the scope of functionality as a specific, maintenance-related IIoT implementation. According to this standard, a Condition Based Monitoring (CBM) system should be composed of various functional blocks: 1) Data Acquisition (DA), 2) Data Manipulation (DM), 3) State Detection (SD), 4) Health Assessment (HA), 5) Prognostics Assessment (PA) 6) Advisory Generator (AG) [9]. This corresponds well to a general IoT system architecture with edge computing, as the implementations of the MANTIS architecture prove [10].

IV. Architecture Model and Components

As Figure 4 suggests, the architecture follows the IEC IIoT [5] reference architecture model in general: we utilize the edge computing paradigm in connection with the gateway mediated pattern; and the MANTIS architecture is also planned for multiple tier levels. However, certain features are added to support additional, maintenance-related tasks, as well.

A. Edge Tier

Within the MANTIS use cases, we can find primarily two main types of edge level devices: closed, fully fledged (i.e., production) sites and standalone devices (e.g., vehicles or outdoor measurement points). These two cases require completely different approaches, and completely different design in the edge-cloud interface.

1) Standalone Devices: In this case, there can be standalone machines, vehicles or outdoors systems that are equipped with one or more sensors. Their communication capability is wireless (i.e., via mobile networks) and therefore limited due to the radio interface capabilities. To these cases, MANTIS offers an alleviated implementation of the edge-cloud interface, still relying on the MIMOSA ontology model.

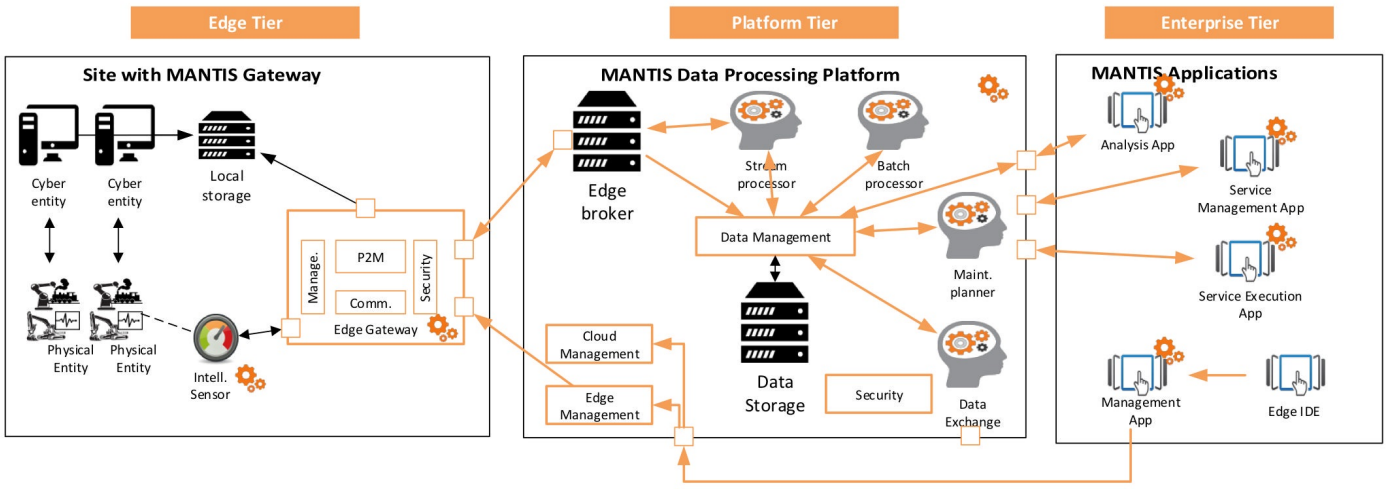


Fig. 4. Overview of the MANTIS reference architecture

In these use cases, the standalone edge devices implement the pre-processing and aggregation algorithms, and only transmit an extract of the high frequency information available locally.

Meanwhile, it is also possible that besides the periodical uploads from these devices, the platform level can still request additional, on demand, temporary data streams from the devices that can help with the verification of a prediction by an analytic module. This measure is also implemented to save bandwidth. However, this might be limited in some use cases, since the nature of the edge device being a low power embedded system operating on battery (e.g., a sensor mote), attached to a machine. It is worth noting here from a development point of view, that these use cases require an extensive measurement phase, where all available data from all possible sources are collected. This "training" phase is required in order to develop the necessary analytic models that will allow us to be able to decide what pre-processing is viable on-board (what information is necessary to transmit and what is redundant).

Furthermore, in these cases, the standalone devices (motes) are also expected to have intelligent functions on their own. These include intelligent sensor management, self diagnostics, and resilience to the unreliable nature of the communications.

2) Cyber-Physical Production Systems: The second typed Edge Tier includes complex cyber-physical systems or legacy production systems ("made smarter") that are are connected through closed, self-contained networks. The primary data sources are these CPSs and we are utilizing their continuous (telemetry) output. Here, we opted for the gateway-mediated edge connectivity design, where a gateway is responsible for communications towards the platform level. These systems are usually not constrained by processing or power limitations, however the communications still need to be efficient, to increase the scalability of the overall platform. Limited pre-processing and analytical modules have to be put locally for at least three reasons. Firstly, process telemetry information in raw is too big of volume to send to any outside platform. Secondly, companies are reluctant to share critical real-time information about their core business, while also legal restrictions might apply [11]. Finally, maintenance personnel is usually located on-site. In a sense, these edge setups are complete on their own.

B. Platform Tier

When realizing the Platform Tier, MANTIS initially employs five modules. These are namely: (i) the edge broker, (ii) multi-purpose (distributed) data storage and management, (iii) stream processors, (iv) batch processors, and (v) human-machine interfaces (HMI). The online analytic functionalites of the MANTIS platform follow the Lambda architecture guidelines and operations. In here, the main data management comprises of the MIMOSA database and its utilities, while the speed and batch layers are dedicated towards the three main maintenance-related objectives, discussed in section II-A. The development of these modules are interated over in two phases: first an offline, manual establishment of the algorithms with expert and data analyst knowledge is carried out, and then the developed solutions are deployed into the online system, taking the specific use case dependent constraints into account.

The modules in the Platform Tier are generally intended to scale well: there can be multiple modules of each type resulting in a large distributed system. In such cases, a big data processing platform is needed for the implementation. A possible implementation builds upon commercial solutions (such as Microsoft Azure [12] or Amazon AWS [13]), but an open source implementation is also possible using for example the Apache ecosystem consisting of the Apache Kafka [14] as the distribution, Apache Storm [15] and Spark [16] as stream- and batch-processors.

The *Edge Broker* is responsible for keeping the direct communication with the edge level devices. It provides translation between the data format used in the edge-cloud interface, and towards the various cloud modules. It also includes the addition of all the required asset information to the upstream data to be MIMOSA compliant. Based on the added information, all the processing and database nodes in the platform can identify and process the inbound data. Moreover, since the edge-cloud interface is not restricted to one implementation, the edge broker implements multiple transport protocols. Therefore, edge devices can communicate via publish-subscribe protocols [17]

(such as MQTT) or request-response type of protocols (such as RESTful HTTP). In a sense, the edge broker is fulfilling the role of an enterprise service bus [18], and here consists of three major components:

1) Protocol facades (e.g., an MQTT broker or an HTTP server, to receive the messages from the edge);
2) Message parser and translator (from the edge-cloud interface to the data distributor feed);
3) Client to the data distributor module (to push the translated message into the various platform modules).

Within MANTIS, the Edge Broker has been implemented in a multitude of ways. One cornerstone of this module is that it might be use case and edge device dependent. The MANTIS platform does not wish to rule out legacy or commercial off the shelves (COTS) implementations for brown field use cases: the cost of deployment for MANTIS is intentionally kept to the minimum. Therefore, the edge broker is modular, and its main purpose is to translate between the various ways of communication formats within the framework. Moreover, since bidirectional communication is expected for some use cases in the edge-cloud interface, the edge broker is addressable by the cloud modules and can send messages (i.e., commands) towards the edge devices, as well.

The *Data Management* system (a.k.a. data distributor) is only required when a large system is being implemented, for scalability and robustness reasons. In smaller deployments, the edge broker can feed the inbound data to other modules directly as well. This module is basically a message oriented middleware on its own. Its purpose is to fetch the inbound data stream from the edge broker, and pass it on to the appropriate modules. One popular solution is the Apache Kafka [14].

The main data storage of MANTIS is based on MIMOSA. A popular implementation of this is a Microsoft SQL database (out of the box), deploying the MIMOSA structure, and a management interface (over RESTful HTTP, developed by MANTIS). This database is used for handling various types of information: from raw sensory data to the scheduled maintenance events. This central database provides hence the historical data (per asset and measurement point) for the analytic modules, and the MANTIS-developed HMI also utilizes it to fetch all information required for the current viewpoint over the system. Therefore, this centralized data storage solution poses a current, but identified bottleneck within the system. However, current efforts are spent to devise a more scalable implementation of MIMOSA to enhance the scalability of the platform, using distributed cloud storage solutions, such as the Apache Hadoop [19].

It is worth noting, that fully fledged production edge systems can also have their own local storage, local MIMOSA instance. This enables easy operation and implementation: every level utilizes its own MIMOSA instance, and when further interaction is needed between levels, it can happen via simple database synchronization, using the same semantics. This is one of the great advantages brought by MIMOSA, besides the implemented standard-compliant domain expertise (operational management) and affiliated information ontology.

The *Stream Processing* functionality is required for several maintenance functions and features, that can be executed in real-time. Such functions include detection and triggering of different types of events, based on simple rules such as thresholds. A typical example is when a measurement exceeds a threshold indicating a failure condition, and further investigation is needed to confirm the failure. Moreover, various Key Performance Indicators (KPIs) for predictive maintenance are also computed on-the-fly by the stream processor. Such KPIs include for example the Remaining Useful Life (RUL) of the main components.

The *Batch processors* are designated to run asynchronous tasks on big chunks of data, usually off-line. Such functions include root-cause analysis, prognosis estimation and possible recalibration of the applied machine learning models. These typically require further information or historical data, fetched from the MIMOSA storage. These processes might be triggered by the stream processors during runtime, and they perform complex tasks that are not needed to be real-time. An example scenario here is the detection of a possible failure: a value in the streaming data passing a threshold initiates a longer (batch) analysis on the system logs, for example looking for an known failure pattern.

C. Enterprise Tier

The Enterprise Tier consists of the following elements.

- *Analysis Applications* provide result dashboards, as well as analysis request HMI for the operators and other experts at the enterprose level;
- *Service Management Applications* enable configuration and tracking of the services provided by the overall architecture in the given domain with all of its applications;
- *Service Execution Applications* support service deployment and execution;
- *Management Applications* enable configuration and tracking of the status for the *platform*, interfacing the Cloud- and Edge management functions at the Platform Tier;
- *Edge IDE* supports the configuration of the Edge Tier equipment and network setup through an Integrated Development Environment.

Beside traditional functionalities of *HMI applications* such as presentation and processing of information, advanced features include explanation and adaptability based on user and application models, as well as knowledge-based systems for decision support. While MANTIS strongly emphasizes autonomy, self-testing and self-adaptation, human role remains one of the important factors in the system operation. The human role is twofold: controlling, which comprises continuous and discrete tasks of open- and closed-loop activities and problem solving which includes the higher cognitive tasks of fault management and planning.

The HMI also benefits from the MIMOSA based implementation of the MANTIS architecture. As all information are entered in the database in a well-defined format, the relevant information can be easily extracted and presented in a unified manner. Furthermore, participating in the distribution process by subscribing to the relevant channels, an efficient HMI can be implemented. It supports decision making by proactively pushing relevant information to the right people at the right

time, by intelligently filtering and summarizing information to prevent information overload through context awareness, by automatically and dynamically scheduling and adapting maintenance plans, thereby keeping the human in the loop at all times.

D. Multi stakeholder interactions

One major work ahead for MANTIS is the realization of multi-stakeholder integration and support for collaborative (maintanance) decision making: external and other corporate internal parties should be able access information tailored for them. One examplary use case is the establishment of the necessary collaboration between the supplier of replacement parts and the service departments, since these have high stakes in the maintenance operations. This requires a service-oriented approach [20].

Multiple Platform or Enterprise Tiers shall be enabled to access information coming from one single production site or edge device, in a controlled way. The same goes vice versa, one edge deployment shall be able to locate and connect to additional (external) services, once local decision making algorithms are deployed. An example case for this might be that a production plant shall be able to inquire replacement part orders if it detects the need for it (based on RUL estimation). All this can be aided by the architecture, so that multiple stakeholders can run-time receive and request information they need, in an asynchronous way. For this matter, the integration of the Arrowhead framework [21] is planned.

V. Conclusions

Proactive maintenance for the CPS domain requires solutions that cover data gathering, storage, processing, feedback and presentation to the human operator. While examples of custom-tailored systems are appearing, this paper presents a generic platform together with specific toolset to cover the problem space. Typical target areas of proactive maintenance include Failure Prediction (FP), calculation of the Remaining Useful Life (RUL), Root Cause Analysis (RCA), among others.

Beside providing an architectural view on data gathering and handling for the given area, the MANTIS architecture for CPS-based proactive maintenance provides solutions for the Egde Tier, for the Platform Tier, and for the Enterprise Tier, as well. The main building blocks of the Edge Tier are physical sensors and actuators, as well as local, edge-level data processing entities. These also communicate with the elements of a system-wide view at the Platform Tier. Depending on the targeted area (i.e., FP, RUL, RCA), stream processing or batch processing entities handle the data and provide meaningful output that either gets feed back to the physical entities as control information, or gets presented towards the Enterprise Tier for further processing or action (e.g., ordering spare equipment, scheduling jobs, visualizing trends, etc.). This architecture already has been successfully utilized in various use-cases from industrial utility vehicles (forklifts) through railway control system to Cyber-Physical Production Systems (CPPSs) [10], as part of the ECSEL MANTIS project [1].

Acknowledgment

This work has been developed with the support of funds made available provided by the European Commission in the scope of ECSEL/H2020 MANTIS Research and Innovation Action (Project ID: 662189). The authors would like to express their gratitude to Michele Albano, and all colleagues and partners in the project.

References

[1] The MANTIS consortium , "The mantis project website," http://www.mantis-project.eu/.

[2] E. Jantunen, U. Zurutuza, L. L. Ferreira, and P. Varga, "Optimising maintenance: What are the expectations for cyber physical systems," in *Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), 2016 3rd International Workshop on*. IEEE, 2016, pp. 53–58.

[3] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of internet-of-things platforms," *Computer Communications*, vol. 89, pp. 5–16, 2016.

[4] ISO, *Condition monitoring and diagnostics of machines – Data processing, communication and presentation*, Standard ISO 13 374, 2012.

[5] Industrial Internet Consortium, "The industrial internet of things reference architecture," 2017.

[6] M. Cengarle, S. Bensalen, J. McDermid, R. Passerone, A. Sangiovanni-Vincetelli, and M. Torngren, "Characteristics, capabilities, potential applications of cyber-physical systems: a preliminary analysis," 2013.

[7] I. E. Commission, *Enterprise-control system integration*, Std. IEC 62 264, 2003–2007.

[8] M. Hausenblas and N. Bijnens, "The lambda architecture website," http://lambda-architecture.net/, 2017.

[9] MIMOSA consortium, "The mimosa project site," http://www.mimosa.org/, 2016.

[10] L. L. Ferreira, M. Albano, J. Silva, D. Martinho, G. Marreiros, G. di Orio, P. Maló, and H. Ferreira, "A pilot for proactive maintenance in industry 4.0," in *Factory Communication Systems (WFCS), 2017 IEEE 13th International Workshop on*. IEEE, 2017, pp. 1–9.

[11] C. Donnelly, "Eu data protection regulation: What the ec legislation means for cloud providers," http://www.computerweekly.com/, January 2015.

[12] Microsoft, "Azure web services," https://azure.microsoft.com, 2017.

[13] Amazon, "Amazon web services," https://aws.amazon.com/products/analytics/, 2017.

[14] Apache Community', "The kafka distributed streaming platform," http://kafka.apache.org/, 2017.

[15] Apache Community, "Storm stream processor," http://storm.apache.org/, 2017.

[16] Apache, "Spark," https://spark.apache.org/, 2017.

[17] E. Curry, "Message-oriented middleware," *Middleware for communications*, 2004.

[18] IBM, "The enterprise service bus, re-examined: Updating concepts and terminology for an evolved technology," https://www.ibm.com/developerworks/websphere/techjournal/1105_flurry/1105_flurry.html, 2011.

[19] Apache, "Hadoop distributed file system," http://hadoop.apache.org/, 2017.

[20] M. Bell, *Introduction to Service-Oriented Modeling*. Wiley and Sons., 2008.

[21] J. Delsing, *IoT Automation: Arrowhead Framework*. CRC Press, 2017.