3

The MANTIS Reference Architecture

Csaba Hegedűs[1], Patricia Dominguez Arroyo[2], Giovanni Di Orio[3], José Luis Flores[4], Karmele Intxausti[4], Erkki Jantunen[5], Félix Larrinaga[6], Pedro Maló[3], István Moldován[7], and Sören Schneickert[8]

[1]AITIA International Inc., Hungary
[2]University of Groningen, The Netherlands
[3]FCT-UNL, UNINOVA-CTS, Caparica, Portugal
[4]IK4-Ikerlan, Arrasate-Mondragón, Spain
[5]VTT Technical Research Centre of Finland Ltd, Finland
[6]Mondragon Unibertsitatea, Arrasate-Mondragón, Spain
[7]Budapest University of Technology and Economics, Hungary
[8]Fraunhofer IESE, Germany

The purpose of this chapter is to describe the MANTIS reference architecture. The generic focus here is on an architecture that enables service-based business models and improved asset availability at lower costs through continuous process and equipment monitoring, aided by big data analysis. This architecture takes into account needs of various industries in the forefront of service-based business and operations models. It also takes into account less mature industrial domains, where improvements in maintenance can be only achieved gradually and consistently. The higher level requirements for the whole project are described by [The MANTIS Consortium, 2018] and further tuned for the architecture in [Jantunen et al., 2016].

3.1 Introduction

The MANTIS proactive service maintenance platform and its associated architecture draws inspiration from the CPS approach. Physical systems (e.g., industrial machines, vehicles, renewable energy assets) operate in an environment, where everything is continuously monitored by a broad and diverse range of intelligent sensors.

This continuous, high resolution monitoring eventually results in massive amounts of data. Systems are characterized, for example, by their usage history, operational conditions, location, movements and other physical properties. These systems and machines form larger collaborative systems-of-systems over heterogeneous networks (e.g., vehicle fleets, photo-voltaic or windmill parks), and hence should be connected via robust communication mechanisms able to operate in challenging (industrial) environments. Here, sophisticated, distributed sensing and decision making functions are performed at different levels in a collaborative way ranging from:

- the local nodes (that pre-process raw sensor data and extract relevant information before transmitting it, thereby reducing bandwidth requirements of communication);
- over intermediate nodes (that offer asset-specific analytics to locally optimize performance and maintenance);
- into cloud-based platforms (that integrate information from ERP, CRM and CMMS systems and execute distributed processing and analytic algorithms for supporting global decision making processes).

For the optimal maintenance of assets, different systems, and stakeholders will have to share information, resources, and responsibilities. In other words, collaboration is required. Such a Collaborative Maintenance Ecosystem will have to be able to reduce the adverse impacts of maintenance on productivity and costs, increase the availability of assets, reduce time required for maintenance tasks, improve the quality of the maintenance service and products, improve labor working conditions, and maintenance performance, increase sustainability by preventing material loss (due to out-of-tolerance production), and help optimizing spare part management.

The overall concept of MANTIS aims to provide a proactive maintenance service platform architecture that allows the precise forecasting of future performance, the prediction and prevention of imminent failures, and should also be able to schedule proactive maintenance tasks. This proactive maintenance service platform will consist of distributed processing chains

that can efficiently transform raw data into knowledge while minimizing the need for transfer bandwidth, as already mentioned in Chapter 1.

Reference architectures provide a template, often based on the generalization of a set of solutions. This is also the case for MANTIS. These solutions may have been generalized and structured for the depiction of one or more architecture structures based on the harvesting of a set of patterns that have been observed in a number of successful implementations. Further, it shows how to compose these parts together into a solution. Reference architectures can be instantiated for a particular domain or for specific projects.

The role of the reference architecture in MANTIS is to provide guidance on how to instantiate an architecture for a particular MANTIS domain or specific MANTIS task, and to ensure consistency and applicability of technologies, interoperability mechanisms, data formats and models, and data analysis tools to be used in the different MANTIS use cases.

Based on requirements and use case descriptions, the reference service platform architecture and overall design needs to address a number of aspects. Important aspects addressed by this chapter are:

- **Interface, protocol, and functional interoperability** ensuring that several cooperating vendors can effectively assemble the complete MANTIS service platform. Includes the need to identify or develop standards for data semantic representation and exploitation;
- **Data validation** ensuring that data analyses are made on data that give clean, correct and useful data information about the system;
- **Distributed data, and information processing, and decision-making** ensuring consistent behavior and avoid contradicting actions, e.g., between local and distributed data analysis and decision making;
- **Information validation** ensuring that created information still is relevant for the system analyzed;
- **System and service level security** ensuring that the system incorporates means to hinder misconfiguration and can be protected from wire-tapping and various attacks;
- **System engineering and re-usability** of defined and existing services;
- **System verification and validation** of the service platform architecture and overall design, covering both functional and non-functional properties.

3.1.1 MANTIS Platform Architecture Overview

The development of a specific implementation of the reference architecture can make use of any of the generalized artifacts, described in this Chapter. All of them help in various ways to avoid having to create a whole reference architecture from scratch, and help to leverage the knowledge and experience that went into the formation and definition of the generalized models, architectures, and patterns.

Adopting a reference architecture within an organization accelerates delivery through the re-use of an effective solution and provides a basis for governance to ensure the consistency and applicability of technology use within an organization. In the field of software architecture, empirical studies have shown the following common benefits and drawbacks from adopting a software reference architecture within organizations [Martinez-Fernandez et al., 2015]:

- improvement of the interoperability of the software systems by establishing a standard solution and common mechanisms for information exchange;
- reduction of the development costs of software projects through the reuse of common assets;
- improvement of the communication inside the organization because stakeholders share the same architectural mind-set;
- influencing the learning curve of developers due to the need of learning its features.

The purpose of the MANTIS ecosystem is to make proactive maintenance possible in a scalable, multi-leveled way. We are targeting CBM: the processes are defined by the ISO 13374 standard [ISO, 2012].

In order to enable maintenance optimization and new business models within MANTIS, appropriate utility services and modules are elaborated and implemented. Within these, data mining and analytic services are created, which are mainly related to these functions [Jantunen et al., 2016]:

- RUL of components: continuous tracking of telemetry (usage) data and estimating how much time the given device or component has left before needs to be replaced;
- FP: the system shall predict based on diagnostic data an inbound failure mode (different to wear-out to be detected by RUL);
- RCA: when an unpredicted, complex failure occurs, the system shall deduct the actual module, the rout caused of the issue;

- MSO: provide a decision making support on better maintenance planning.

To facilitate the requirements and business goals, the MANTIS ARM consists of five elements from high-level design to implementation of the architecture in various use cases:

- Reference Model: a reference model is an abstract framework for understanding significant relationships among the entities of some environment;
- Reference Architecture: provide a template solution for the architecture (aka. architectural blueprint) for a particular domain;
- Feature model: Introduces key concepts to characterize common and varying aspects in the architectures to be derived from the reference architecture;
- Guidelines: discusses how the provided models, views and perspectives are to be used;
- Reference applications: show the diversity of the included solution variants, and thus illustrate architecture signification features and related design decisions.

The approach for architecting MANTIS use cases follows the principle of architecting for concrete stakeholder concerns (based on Architecture Drivers). These stakeholder concerns will drive the eventual architecture design, which is based in the approach follow by the SPES consortium [Pohl et al., 2012]. This approach suggests to start by delineating system and its context, then to continue with the functional decomposition of the system. The next step is the software realization. The final steps consider the hardware realization of functions and the deployment of software entities, as depicted in Figure 3.1.

3.2 The MANTIS Reference Architecture

As discussed in Chapter 2, many of the requirement categories are 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 addressed by MANTIS on an implementation level (since being a reference architecture model), although, when designing an installation, we have to keep in mind that the final, integrated systems has to cover these as well. These categories included (i) data handling, (ii) event handling, (iii) guarantee-related, and (iv) security issues.

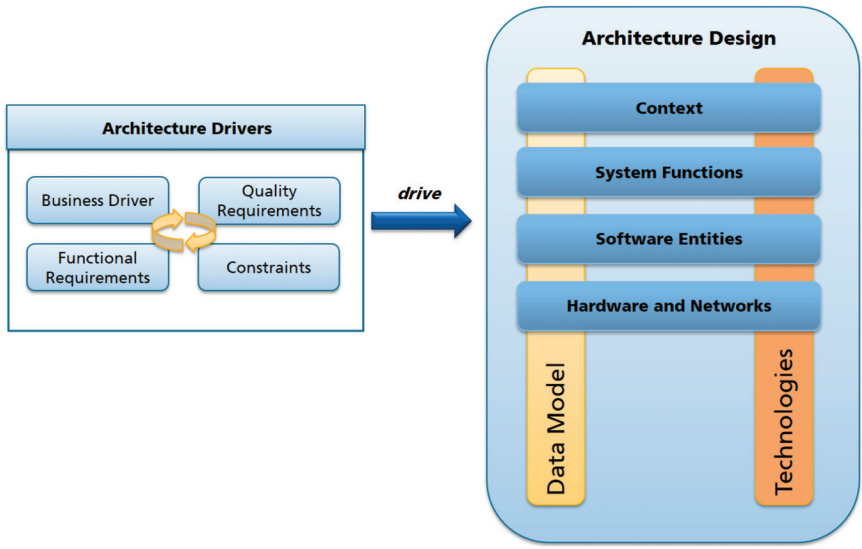

Figure 3.1 Mantis architecture construction approach.

The most important implementation-related requirements are aimed towards scalability and 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 and maintenance operations personnel. It is worth noting that the scope of MANTIS platform architecture does not include or target the actual life-cycle management of the devices. To do that, MANTIS relies on the already existing corporate systems, and resources. However, these interactions with external systems is planned and designed into the framework via standardized secure communications between platform modules using the interoperability guidelines set [Di Orio et al., 2018].

3.2.1 Related Work and Technologies

Every novel result is based on previous work, which acts as background for the novel advances. In fact, background information lays the foundations for the MANTIS architecture, which is built over novel and existing technologies that are composed to allow for the MANTIS maintenance strategies. Moreover, other related work acts as reference and comparison for the MANTIS architecture. This section describes this plethora of information to support the description and discussion on the architecture.

3.2.1.1 Reference architecture for the industrial internet of things

CPSs are nowadays built together within some form of IoT architectures. A "usual" IoT application employs various *things* collecting enormous amounts of data from a number of places and sending them to an *IT cloud* for a specific purpose. A survey of 39 IoT platforms [Mineraud et al., 2016] concluded in a generic architecture and common characteristics of IoT platforms. Figure 3.2 depicts a generalized commercial IoT platform in its fullest form, and two possibilities are shown. The various IoT modules and services can be deployed on local premises – or within a global IT cloud, depending on the restrictions made by the use case.

The generic modules in such a platform are the following [Mineraud et al., 2016]:

- Sensor or actuator nodes, i.e., "motes" or "things" 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 on their own;
- 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");

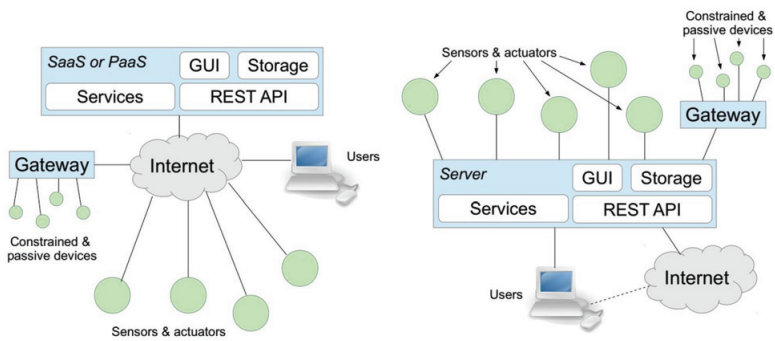

Figure 3.2 A generalized IoT framework [Mineraud et al., 2016].

- Graphical interfaces for operators to manage the system and validate the output (i.e., "Business Insights").

The data gathering and processing viewpoint shown by Figure 3.3 corresponds to that of the IIoT reference architecture proposed by the IIC [Industrial Internet Consortium, 2017]. However, three additional architecture patterns are proposed 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 fundamental value added of the IIoT approach: enhancing *"legacy" production systems* by "making them smart" with additional, usually non-invasive components. This additional device (i.e., gateway) shall utilize e.g., the management interfaces of the machines, and represent the functionalities provided there using IP-based interfaces. Nevertheless, this results in the physical machines being connected to the network, their operations offered as "services" – hence creating CPSs [Cengarle et al., 2013] out of them.

Therefore, we also have to create a logical space that implements a common schema, while it also has to provide a "language" used in the communications between endpoints (i.e., translation between various data description ontologies into one understanding). Such a logical data bus design

Figure 3.3 The industrial internet of things reference architecture.

pattern hence supports communication between applications and devices: semantics and translation are the basis for interoperability within MANTIS, and in IIoT, in general.

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). In here, we are tapping out information from the communications within the (real-time) control loops between the given CPSs (cf. ISA95 systems [International Electrotechnical Commission, 2003–2007]). This way, we are collecting mostly process telemetry, then aggregating and preprocessing it locally. This is usually supported by an *application gateway* that provides the connectivity: it bridges to a WAN towards the platform level(s). It also acts as an endpoint for the WAN, 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, however, is that this way, we are breaking down the complexity of IIoT systems, so that they may scale up both in the numbers of managed assets as well as in networking.

The access network enables connectivity for data and control flows between the edge and the platform tiers. It may be a corporate network, or an overlay private network over the public Internet or a 4G/5G network.

The Platform Tier receives the streams of telemetry data from the Edge tier. It is also executing the control commands coming from the Enterprise Tier, and may forward some of these commands to the Edge Tier in a cloud-to-device manner. It consolidates and analyzes the data flows from the Edge Tier and other systems. It provides management functions for devices and assets (e.g., Over The Air firmware updates for the application gateways). 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 any generic IoT platform.

Meanwhile, the Enterprise Tier receives the processed data flows (i.e., business insights) coming from the Edge devices towards Platform Tiers. It might also issue control commands to the Platform and Edge Tiers. This tier is the main beneficiary of the IIoT system. However, the utilization of a well-built MANTIS platform is also not an easy task on the corporate side either. The issues are presented and tackled in Chapter 8.

3.2.1.2 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. To do so, there are many

paradigms, software stacks (both open source and commercial), consultant firms. However, in general, the data processing usually follows the same logic. Within MANTIS, the Lambda architecture [Hausenblas and Bijnens, 2017] is considered, however, there are many, similar "competitors" of it as well [Kappa, 2018].

According to the generalized Lambda architecture pattern [Hausenblas and Bijnens, 2017] 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 3.4 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

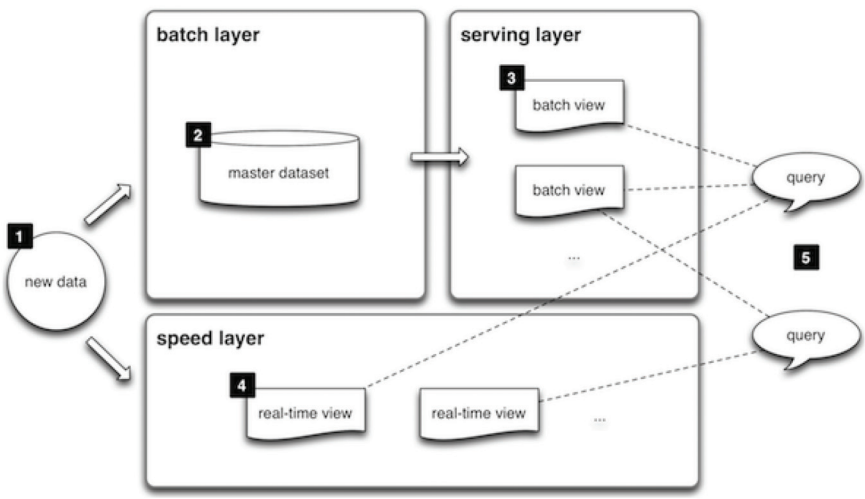

Figure 3.4 The Lambda data processing architecture.

any IIoT application. Many commercial products and platforms support these operations.

The batch layer has two major tasks: (a) managing historical data; and (b) recomputing results such as machine learning models. Specifically, the batch layer receives arriving data, combines it with historical data and recomputes results by iterating over the entire combined data set. The batch layer operates on the full data and thus allows the system to produce the most accurate results. However, the results come at the cost of high latency due to high computation time. The speed layer is used in order to provide results in a low-latency, near real-time fashion. The speed layer receives the arriving data and performs incremental updates to the batch layer results. Thanks to the incremental algorithms implemented at the speed layer, computation cost is significantly reduced. 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).

3.2.1.3 Maintenance based on MIMOSA

OSA-CBM has developed an system architecture for condition-based maintenance, i.e., a way to enhance the modularization of different vendor systems, while not locking customers into a single-source solution. The MIMOSA has since the middle of the 1990's hosted open conventions for information exchange between plant and machinery information systems, namely a way to enhance, amongst other things, the compatibility issue between different vendor products [MIMOSA consortium, 2016].

The OSA-CBM and MIMOSA are two major standard organizations and they claim that an accepted non-proprietary open system architectural standard is important, since it would bring an improved ease of upgrading system components, a broader supplier community, more rapid technology development, and reduced prices [Lebold and Thurston, 2001].

One of MIMOSA's most valuable contributions are the development of a CRIS. It is a relational database model for different data types that need to be processed in a CBM application. The system interfaces are defined according to the database schema based on CRIS. The interfaces' definitions developed by MIMOSA are an open data exchange convention to use for data sharing in today's CBM systems. In addition, defined by MIMOSA Cris is also MIMOSA's OSA-EAI, which provides an open exchange standard,

for technology types, in key asset management areas, such as asset register management, work management, diagnostic and prognostic assessment, vibration and sound data, oil, fluid and gas data, thermographic data and reliability information.

Besides supporting all the data needed for a CBM application it also considers for instance the CMMS handling, to be precise work management of a maintenance department. The structure of data in a relational database is predefined by the layout of the tables and the fixed names and types of the columns, which is the case of the MIMOSA CRIS database model. In addition, during 2012 the OSA-EAI V3.2.3 released a complete UML model and XML schema implementation called CCOM-ML for the CCOM, in addition to continued support and updates for CRIS.

Within MANTIS, MIMOSA [MIMOSA consortium, 2016] is providing the common understanding and data ontology between partners and applications (Figure 3.5). 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 data bus [Industrial Internet Consortium, 2017], while remaining loosely coupled. MANTIS has developed a full stack of message models extending the MIMOSA ontology

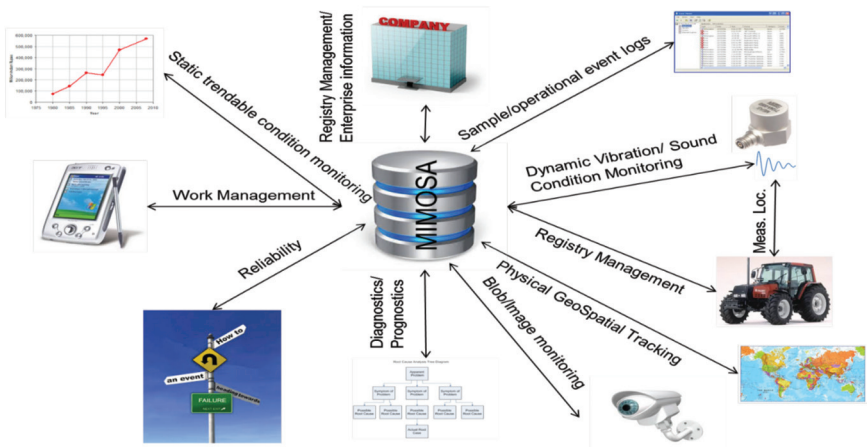

Figure 3.5 MIMOSA data model diagram [MIMOSA consortium, 2016].

that are designed to facilitate communications between edge and cloud, and also between the various cloud modules.

Moreover, MANTIS follows the ISO-17359 standard in terms of the scope of functionality as a specific, maintenance-related IIoT implementation. According to this standard, a CBM system should be composed of various functional blocks, which then corresponds well to a general IoT system architecture with edge computing, as the implementations of the MANTIS architecture. In Figure 3.6 can the three parts be visualized against the OSA-CBM architecture.

3.2.2 Architecture Model and Components

As Figure 3.7 suggests, the architecture follows the IEC IIoT [Industrial Internet Consortium, 2017] reference architecture model in general, in the sense of using 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.

3.2.2.1 Edge tier

Within the MANTIS use cases, there are 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.

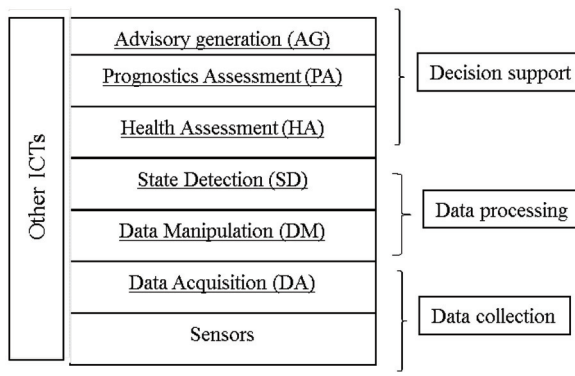

Figure 3.6 A three-part CBM architecture in the light of the OSA-CBM [Lebold and Thurston, 2001].

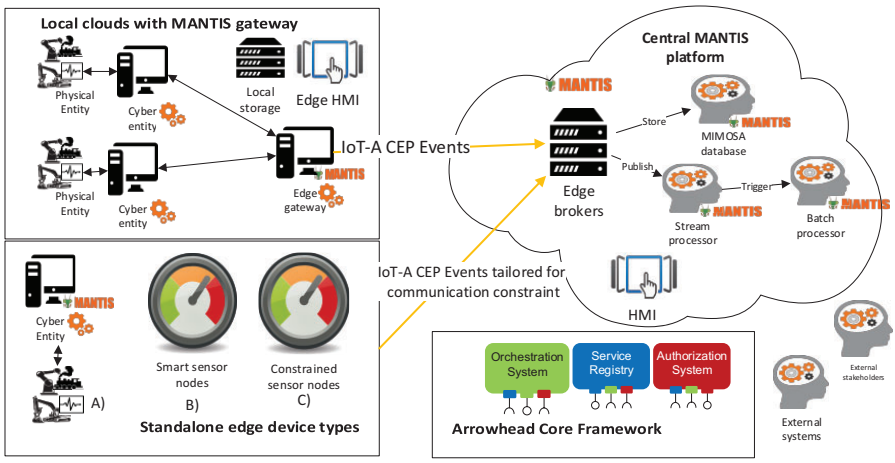

Figure 3.7 Overview of the MANTIS reference architecture.

• *Standalone Devices*

In this case, there can be standalone machines, CPSs, vehicular system or other outdoors systems that are equipped with various sensors. Their communication capability is mostly wireless (i.e., via mobile networks) and therefore limited due to the radio interface capabilities. To these cases, MANTIS proposes an alleviated implementation of the edge-cloud interface, still relying on the MIMOSA ontology model. In these use cases, the standalone edge devices implement some pre-processing and aggregation algorithms, and only transmit an extract of the high frequency information available locally in order to save traffic.

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. This might help with the verification of a prediction provided 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 extensive training and familiarization phases, where all available data from all possible sources are collected. This phase is required in order to develop the necessary local analytic models that allows 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. MANTIS does not propose limitations on these matters, but provides guidelines based on the lessons learned in the sensor developments of the project.

• *Cyber-Physical Production Systems*

In many of the cases, the Edge Tier includes complex CPSoS, based on legacy production machines ("made smarter") that are are connected through closed, self-contained networks. The primary data source consists in these CPSs, by means of their continuous (telemetry) output. Within MANTIS, building the automation based on the CPS context is out of scope.

Here, the choice fell on 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. Rather the communications need to be efficient for different purposes: to increase the scalability of the overall platform. There was also the need to design various pre-processing and analytical modules that have to be put locally for at least three reasons. Firstly, raw process telemetry information is too much 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 [Donnelly, 2015]. Finally, maintenance personnel is usually located on-site. In a sense, therefore, these edge setups are complete on their own.

3.2.2.2 Platform tier

When realizing the Platform Tier, MANTIS initially employs five modules. These are:

- the edge broker;
- multi-purpose (distributed) data storage and management;
- stream processors;
- batch processors;
- HMI.

The speed and batch layers are dedicated towards the three main maintenance-related objectives. The development of these modules are iterated over two phases: first an off-line, manual establishment of the algorithms with expert and data analyst knowledge is carried out, and then the developed solutions are deployed into the on-line 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. Possible implementations build on commercial solutions, but open source implementations can also be applied. Within MANTIS, the following platforms have been used, without being exhaustive:

- Microsoft Azure [Microsoft, 2017];
- Amazon AWS [Amazon, 2017];
- Apache ecosystem (Kafka [Apache Community, 2017], Storm [Apache Community, 2017] and Spark [Apache, 2017]);
- Wapice IoT-Ticket [Wapice, 2018].

The *Edge Broker* is responsible for keeping the direct communication with the edge level devices and gateways. It provides translation between the data format used in the edge-cloud interface, and within the platform level. Its primary purpose is to forward the inbound data 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 usually implements multiple transport protocols. Therefore, edge devices can communicate via publish-subscribe protocols [Curry, 2004] (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 [IBM, 2011], and here consists of three major components:

- Protocol facades (e.g., an MQTT broker or an HTTP server, to receive the messages from the edge);
- Message parser and translator (from the edge-cloud interface to the data distributor feed);
- 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 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, using the interoperability messaging schema of 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 (or data distributor) is needed when a large system is being implemented, to ensure scalability and robustness. In smaller deployments, the edge broker can forward the inbound data to other modules in the cloud, directly as well. Basically, this module is a message oriented middleware on its own. Its purpose is to collect the inbound data stream from the edge brokers, and build a data pipeline towards the other modules in the platform. One popular solution for data distribution is the Apache Kafka [Apache Community, 2017]. Section 3.3 further discusses the issues and design decisions to be made regarding data management and collection tasks necessary to build such IIoT big data systems as MANTIS.

The main data storage and everything connected to data descriptions within MANTIS is based on MIMOSA. The reference implementation of this domain ontology is provided in a Microsoft SQL database, deploying the MIMOSA structure. MANTIS has developed a RESTful HTTP interface for the database as well. This database is used for handling various types of information: from raw sensory data to the scheduled maintenance events, as discussed in section 3.2.1.3. This central database provides storage of the historical data (per asset and measurement point) for the analytic modules. The MANTIS reference HMI solutions also utilize it to fetch all information required for the overview of the system.

It is worth noting, that fully fledged production edge systems (i.e., CPPSs) can also have their own local storage, local MIMOSA instance. This enables easy operation and implementation: every level/tier 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 ontologies.

The *Stream Processing* functionality is required for several maintenance functions and features, that can be executed in real-time. Such functions include the 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 (hence fault prediction and root cause analysis). Moreover, various KPIs for predictive maintenance are also computed

on-the-fly by the stream processor. Such KPIs include for example the RUL of the main components.

The *Batch processors* are designated to run asynchronous tasks (such as machine learning jobs) on big bulks of data. Such functions here include training root-cause analysis, prognosis estimation on historical data and possible recalibration of the applied machine learning models when sufficient new information has been collected. These typically require further information or historical data, fetched from the MIMOSA storage. These processes might be triggered by the stream processors during runtime or run periodically, 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 (i.e., batch) analysis on the system logs, for example looking for an known failure pattern beforehand.

3.2.2.3 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 enterprise 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.

The Enterprise tier provides the usual features of *HMI applications* such as presentation and processing of information, and moreover it adds mechanisms for explanation and adaptability based on user and application models. Therefore, these are knowledge-based systems for decision support as well. While MANTIS strongly emphasizes autonomy, self-testing, and self-adaptation, the human role remains one of the important factors in the system operation. It is however 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. These issues are further described in the next Chapters,

as considerations are made for the generic parts (e.g., HMI) and installation-specific issues as well.

The HMI also benefits from the MIMOSA based implementation of the MANTIS architecture. As all information are stored in a 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.

3.2.2.4 Multi stakeholder interactions

One major issue tackled by MANTIS is the realization of multi-stakeholder integration and support for collaborative (maintenance) decision making: external and other corporate internal parties should be able access information tailored for them. One exemplary 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, as described in [Jantunen et al., 2018]. This requires a service-oriented approach [Bell, 2008].

Multiple Platform or Enterprise Tiers are 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 [Delsing, 2017] is proposed since it also supports other capabilities that are useful for advanced maintenance operations, such as Quality of Service [Albano, 2017].

3.3 Data Management

Since MANTIS is proposing big data based analytic solutions to solve Maintenance 4.0 problems, the data storage solutions are essential to discuss

and design into the framework's core architecture. Cisco has forecasted that by 2020, 92 percent of the workloads will be processed by cloud data centers of which 68 percent will be in public data centers [Cisco, 2016]. Within project MANTIS, Big Data are collections of data entries having the following characteristics [Munshi and Yasser, 2017; Laney, 2001; Assunção et al., 2015] (while adding the newest and oriented interpretation [Fan and Bifet, 2013; Grover and Kar, 2017] dimensions as well):

- *Volume*: big data implies enormous volumes independently of the data source: machine or environmental sensors, event logs, external data sources;
- *Variety*: the diversity in source and format of the data collections. Although data repositories or processing may allow a restricted amount of heterogeneity, it is safe to include variety as one of the implicit features of big data as a whole;
- *Velocity* (i.e., data generation rate): Notwithstanding that collections and repositories may be static at a certain point, data is generated over time and that is what is referred to as velocity. Data might also be geographically distributed to make geographic static classifications, however such static representations of geographic distribution in an isolated or discrete point in time are more on the statistical side than in the big data spectrum;
- *Variability* in the units, data structures and formats that hold an equivalent representation of a measurement, state or data entries from different sources. Data transformations have to guarantee that input as output are equally faithful to the information they account for;
- *Value* is the engineering process behind that converts any piece of information into a mercantile asset that has enterprise value;
- *Veracity* makes reference to the quality of data. Raw data is often preprocessed to fit a particular data process. The veracity of the data refers to the appropriate handling of data formats and preprocessing transformations. Data transformations should always keep the relative real representation of the truth status they represent.

For data to be turned into an economic asset, so called Big Data, engineers, architects and scientists are involved in the data gathering and engineering processes that collect and convert the terabytes of information into meaningful added value. One of the well recognized obstacles that prevents potential useful data from being exploited is that 60% to 80% of data mining efforts is strictly dedicated to the preparatory work. This is yet another

motivation to carefully study the needs of a data exploitation infrastructure and put attention to data engineering process before turning data into insights and building a MANTIS (or alike) architecture deployment.

3.3.1 Data Quality Considerations

Thorough the data life cycle, data transformations and data source integration processes, original data entries might suffer from inadequate handling, leading to misleading data analysis results. Some highlighted threats to data integrity are found in the literature [Singh et al., 2010].

Completeness here makes reference to whether data values are missing or that some data points are noisy. Besides that, completeness is a data quality issue that goes beyond the strict integrity of the data entries themselves. It also makes reference to the context information necessary to make adequate interpretations to the data. Meanwhile, validity refers to the correctness and reasonableness of data as the capability to faithfully represent the piece of reality it describes. As a title of example, it implies that data is describing reality in a scientifically accepted manner. Accuracy is the exactness with which data represents the real world. The concern here is whether a specific piece or collection of data or reading "makes sense" in comparison with the actual state of what originates that data. Meanwhile integrity: this entails in the broader sense the lack of corruption of data through the entire life cycle.

3.3.2 Utilization of Cloud Technologies

Cloud services do not necessarily have to be outsourced to third parties entirely and in industrial cases they are not. Within an organization, cloud services can be provided internally, as private clouds; as shared, as hybrid clouds or fully hosted in a third party infrastructure, as commercial solutions. This classification entails the following [Lenk et al., 2009; Hashem et al., 2015]:

- Private/Local clouds: all services, infrastructure, platform and software on which a solution runs, are hosted entirely by the party that exploits the solution. This does not imply that all the services in the system are proprietary solutions, there can be commercial, licensed products (services) hosted in a private environment;
- Hybrid clouds: the solution is partially hosted by a third party service provider and partially hosted by the same entity that exploits the service. The distribution of who hosts what does not affect the classification;

- Public or Commercial clouds: these are fully hosted by a service provider that is a different entity than the entity exploiting or consuming the service. In this case the differentiation between service provider and user/consumer is complete.

It is possible, however, that a particular implementation has a service provider to fully host one of the layers of a system architecture (i.e., infrastructure, platform, software), in which case, the specific terminology to refer to the cloud hosting scheme can be used differently for each of those layers.

MANTIS promotes the utilization of distributed data storage solutions, regardless on how the solution shares its resources between the various types of CSP and on what level it is utilizing a COTS technology (i.e., provided infrastructure, software or platform "as a Service" [Bermbach et al., 2017]). The interoperability issues and how they can be tackled the MANTIS way when using different solutions and products in the data pipelines is presented in Section 3.4.

3.3.3 Data Storages in MANTIS

The *Central Data Storage* is the main storage system where all maintenance related data is collected. It should be a scalable, possibly cloud based. Its implementation should be possible using both open source tools (e.g., Apache big data framework) or vendor specific (like Microsoft Azure). The adoption of MIMOSA (see Section 3.2.1.3) is recommended in the MANTIS architecture.

High granularity measurement data is usually not needed to be uploaded to the central database. However, *local databases* are maintained (in a CPPS for example), and in case when needed, the data can be accessed from the local database. Such high granularity measurement data can be for example raw vibration measurement data.

Although, in some cases local storage is not practical or it is not possible. However, diagnostics and prediction would benefit of the high granularity data. In these cases, it can be considered the possibility of ad-hoc data request in between the tiers of the MANTIS architecture (e.g., between edge-platform, where the CPS performs a high granularity measurement and provides the data as a response).

The MANTIS reference architecture proposes to encapsulate the actual storage technology behind a respective data management facade in order to mitigate the impact of technology changes of the data storage on the data analysis functions. There are several other databases that may be needed

for the main functions of MANTIS (i.e., RUL, RCA, FP and MSO). Such databases include for example:

- Environmental database (e.g., weather): to complete the diagnostics data environmental conditions may be taken into consideration;
- Component stock database at subcontractors: needed for efficient planning;
- Other examples: Safety/regulatory databases, etc.

Scalability is the service property of the system to adapt to larger workloads (storage or cloud service) such as to "serve X% more requests when deployed on X% more resources" [Bermbach et al., 2017]. This is at least the general concept of scalability, which is also known as vertical scalability.

However, another aspect of scalability concerns the way a system is able to accommodate new functional requirements over time. This characteristic has been given the name of horizontal scalability, it played an important role in the evolution from RDBMS to NoSQL paradigms, with a lower threshold for system adaptation in the latter.

It is also worth noting that many enterprises are not only migrating their database systems to the cloud but also shifting their original relational databases to non structured NoSQL databases. One of the main reasons to adopt this new paradigm is the ability of the services in the digital system to handle real-time data, with high agility and flexibility. The problem is well posed in [Hecht and Jablonski, 2011]: "In the past SQL databases were used for nearly every storage problem, even if a data model did not match the relational model well. The object-relational impedance mismatch is one example, the transformation of graphs into tables another one for using a data model in a wrong way. This leads to increasing complexity by using expensive mapping frameworks and complex algorithms".

3.3.4 Storage Types

MANTIS relies on already existing data sources, or promotes the development of new data collection and storage if legacy sources are insufficient for the targeted analytic services.This section introduces some of the most popular storage solutions classified in a taxonomy of storage infrastructures. Although there are more technologies available both commercial and open source, the following have been researched to take in consideration for possible use. In this section, the following database types are considered for implementation of the MANTIS platform:

- Traditional SQL databases: Microsoft SQL, MySQL, etc;
- Big Data File Systems: Google File System, Hadoop File System, Disco File System;
- NoSQL: Key-Value, Column-Oriented, Documents, Graph.

3.3.4.1 Big data file systems

Big data file systems in this section are easily scalable and can support a variety of data formats, and databases in both structured and unstructured data models. These are the most widely spread and flexible data storage systems, also the most rudimentary in terms of pre-packaged functionality.

The Google File System is a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients [Ghemawat et al., 2015]. A Google File System cluster consists of a single master and multiple chunkservers and is accessed by multiple clients. Chunkservers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range.

The HDFS [Apache, 2017] is also a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.

Disco Distributed Filesystem (DDFS) [Nokia, 2017] provides a distributed storage layer for Disco. DDFS is designed specifically to support use cases that are typical for Disco and MapReduce in general: Storage and processing of massive amounts of immutable data. This makes it very suitable for storing, for instance: log data, large binary objects (photos, videos, indices), or incrementally collected raw data such as web crawls. Although DDFS stands for Disco Distributed filesystem, it is not a general-purpose POSIX-compatible filesystem.

3.3.4.2 NoSQL databases

In line with what the acronym indicates (Not Only SQL) NoSQL data storage systems integrate structured and not-structured or semistructured data structures. NoSQL is becoming increasingly popular for its compatibility with the object oriented programming paradigm that many

application use nowadays [Assunção et al., 2015]. NoSQL technologies avoid the architectural friction called Impedance Mismatch [Sahafizadeh and Nematbakhsh, 2015] between object oriented data generation at edge and hard-shaped relational database information storages. In here, the data types under consideration can be enumerated as (i) key-value, (ii) column-oriented, (iii) document-oriented, (iv) graph-oriented and (v) time-series.

Cassandra [Apache, 2017] is an open source distributed storage for managing key-value typed data. The properties mentioned in [Han et al., 2011] for Cassandra are the flexibility of the schema, supporting range query and high scalability. Cassandra, among others, sadly has the potential for denial of service attacks because it performs one thread per one client and it does not support inline auditing [Noiumkar and Chomsiri, 2014].

Other key-value typed database implementations are the Voldemolt[1], Redis[2] and DynamoDB[3].

Meanwhile, HBase[4] is an open source column oriented database modeled after Google big table and implemented in Java. Hbase can manage structured and semi-structured data and it uses distributed configuration and write ahead logging.

HyperTable[5] is also an open source high performance column oriented database that can be deployed on HDFS. Hypertable does not support data encryption and authentication [Noiumkar and Chomsiri, 2014]. Eventhough Hypertbale uses HQL which is similar to SQL, it has no vulnerabilities for the injection [Noiumkar and Chomsiri, 2014]. Additionally, no denial of service vulnerability is reported to work against Hypertable [Noiumkar and Chomsiri, 2014].

MongoDB[6] belongs to the third category here, namely it is a document-based database. It supports complex datatypes and has high speed access to huge data [Han et al., 2011]. All data in MongoDB is stored as plain text and there is no encryption mechanism to encrypt data files.

There are also databases that are tailored to store time series. In here, arrays of numbers are indexed by time (a range of datetime). In some fields these time series are called profiles, curves, or traces. A time series of stock

[1]Project Voldemort. A distributed database. Online: http://www.project-voldemort.com

[2]Redislabs. Redis. Online: http://redis.io/

[3]Amazon Web Services, Inc. Amazon DynamoDB. Online: http://aws.amazon.com/dynamodb

[4]Apache HBase, Online: https://hbase.apache.org/

[5]HyperTable. Online: http://www.hypertable.org

[6]MongoDB. Online: https://www.mongodb.com/

prices might be called a price curve. A time series of energy consumption might be called a load profile. A log of temperature values over time might be called a temperature trace. Popular implementations are eXtremeDB[7] and Graphite[8].

Other document-typed NoSQL database implementations are the CouchDb and DynamoDB[9]. Meanwhile, Neo4J[10] is an open source graph database, for example.

3.4 Interoperability and Runtime System Properties

The MANTIS approach for interoperability specifications and guidance definition builds up on the main assumption that the identification of a reference model for the interoperability of CPS systems cannot be established without any link to the concrete, instantiated architectures. Therefore, in order to extract the main requirements and models for interoperability and its level of application, the steps presented in Figure 3.8 have been followed.

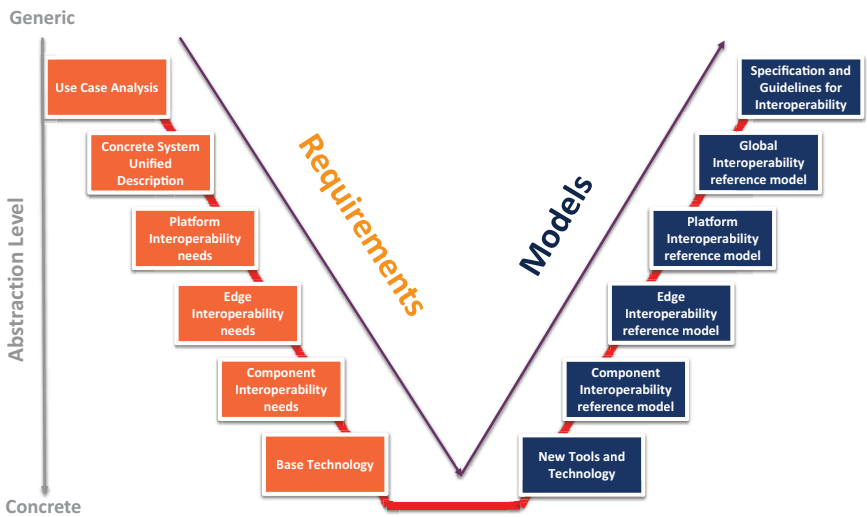

Figure 3.8 MANTIS interoperability proposed approach.

[7] eXtremeDB. Online: http://www.mcobject.com/extremedbfamily.shtml
[8] Graphite. Online: https://graphiteapp.org/
[9] DynamoDB. Online: http://aws.amazon.com/dynamodb
[10] Neo Technology, Inc. Neo4j. Online: http://neo4j.com

These steps provide the interoperability requirements that are used to create reference models for interoperability, i.e., to define specification and guidance to respond to the main interoperability requirements. The main steps of the proposed approach are the following:

- Use Case Analysis: characterization of the use case concrete architecture in which the MANTIS platform will be integrated;
- Concrete System Unified Description: unique description of the overall system, i.e., concrete use case architecture plus MANTIS platform;
- Platform Interoperability needs: identification of the interoperability issues at platform;
- Edge Interoperability needs: identification of the interoperability issues at edge level;
- Component Interoperability needs: identification of the interoperability issues at component level;
- Base technology: identification of the base technologies.

In here, various reference models for interoperability are taken into account and utilized. It is necessary to identify elements on the tools and technology level that could potentially help the integration of the MANTIS platform within the use case ecosystems. Interoperability needs to be handled on four different levels that are component, edge, platform, and on the global level as well.

The component level focuses on how physical entities should be virtualized, i.e., how physical entities can be "cyberized" in terms of the functionalities and/or services that they are able to provide or in other words how to create MANTIS-enabled CPS. The edge level issues revolve around a set of physical entities belonging to the same local system logically represented by a LAN. At this level, the data extracted from physical entities is used to model and analyze the behavior of the system. The edge level also includes a sub-level that is the component level where physical entities are analyzed singularly. At the machine and component sub-level the data extracted from a physical entity is used to model and analyze the behavior of a physical entity singularly.

The platform level needs to describe the information exchange and data integration in the cyberspace. At this level, the data coming from the edge level is organized in order to be processed by MANTIS digital artifacts, i.e., integration between digital artifacts that are responsible to analyze the data

provided by CPS located at edge level. Moreover, the global interoperability reference model needs to provide a unique model that is the confluence of the platform, edge and component interoperability reference models.

3.4.1 Interoperability Reference Model

Inspired by IoT-A [IoT-A Reference Architecture Model, 2018], a MANTIS-ARM has been created to provide the cornerstone for designing, developing and deploying MANTIS-enabled solutions.

The reference architecture provides views and perspectives on distinct architectural aspects that provide the foundation for building compliant architectures. A perspective here defines a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views [International Electrotechnical Commission, 1993]. Therefore, the interoperability perspective is something orthogonal to the several other views defined within the MANTIS reference architecture building blocks: interoperability is considered in all tiers of the architecture.

Considering a complex system (e.g., industrial production system) it is composed of a huge number of machines and their related components. Thus, the machine or component level comprises of physical entities that are part of the same functional unit. At machine and component level the main interoperability issues are related to

- the definition of the granularity level for CPSs;
- the design and development of communication library for extracting raw data from physical entities and appending it into the cyber entity.

The edge level comprises physical entities that belong to the same local network and functional area. The topology and the intrinsic characteristics of the edge level strictly depend on the particular architectural pattern that is used for designing the MANTIS platform. As a matter of fact, the edge level can be as simple as an elementary gateway that delivers data to the platform level (where the intelligence is installed and deployed) or as complex as a network of digital artifacts that provide advanced edge data analytics and knowledge generation functionalities as well as transmission of the data to the platform level for more accurate and resource consuming tasks. In both of the cases, the main interoperability issues are related to:

- the integration of the CPS entities into the edge: It implies the virtualization of the physical entities into CPS (component level interoperability) to be integrated within the edge;
- data extraction, translation and pre-processing from the available edge nodes;
- provisioning of the data extracted from the CPSs (i.e., edge nodes) to the platform level.

The platform level receives, processes and forwards commands from/to the edge level. It provides more complex and resource consuming data analytics and knowledge generation functionalities wrapped into digital artefacts than the edge level. At platform level data received from the edge level are organized according to a common ontology and/or data model that supports the data flow and exchange between the digital artefacts. Therefore, at platform level there are two main interoperability issues. The first one is concerned with the semantic data representation and exchange, to allow the digital artefacts of data analyitics to process the data. The second interoperability issue is on how to represent the knowledge models generated by the digital artefacts of data analytics, in order to enable the usage of such models back to the edge level for local control.

Finally, there are several interoperability issues that are orthogonal to the considered interoperability levels, i.e., models, guidelines and specifications that can be applied to all the interoperability levels without any restriction. These topics reveal interoperability issues that are related on how CPS and, more in general, digital artefacts are connected together. This includes the definition of the communication protocol and message exchange pattern to use in both edge and platform levels and the definition of an ontology of events to support systems interactions at both edge and platform levels.

3.4.2 MANTIS Interoperability Guidelines

The MANTIS Interoperability Guidelines have been structured into three main parts:

- Conceptual integration (modeling and design stage): it is focused on concepts and their relationships, models and meta-models. It provides the modeling foundation for systemizing the relevant interoperability aspects for the specific application domain;
- Application integration (guidance to instantiate the models): it is focused on methodologies, guidance and patterns to support the design and development of their own MANTIS concrete instantiations;

- Technical integration (specific implementation and integration): it is focused on technical aspects related to the networking (protocols, connectivity, etc.), hardware (CPU/memory power preferably low-cost and low-power consumption) and more in general to integration of heterogeneous data sources.

3.4.2.1 Conceptual and application integration

The model for conceptual integration (see Figure 3.9) has been created following a model-driven development approach to enable the design of interoperable and interconnected CPS-populated systems. It starts with the definition of a domain model (see Figure 3.10) that is aimed to capture the essence of the CPS while enabling the specification of the services and interfaces that the CPS must provide. The domain model is then complemented with the semantic data representation and exchange model and the system interaction model. The former is aimed to define and specify the structure of all the data and/or the information handled by CPS at the network level. The latter is aimed to define and specify the relevant events produced/consumed within the MANTIS platform, as well as, the distinct patterns for system interactions.

Architectural aspects are related to the use case specific, concrete architectural pattern used to design the MANTIS platform. Interoperability

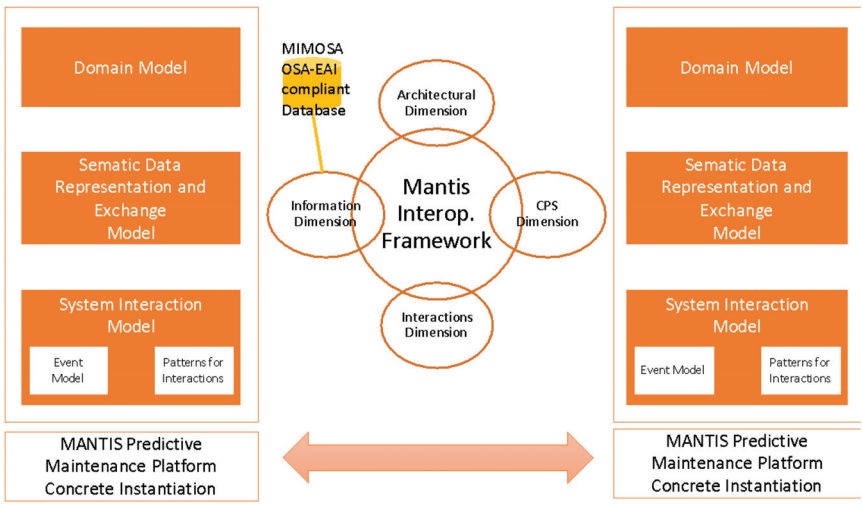

Figure 3.9 Model for conceptual integration.

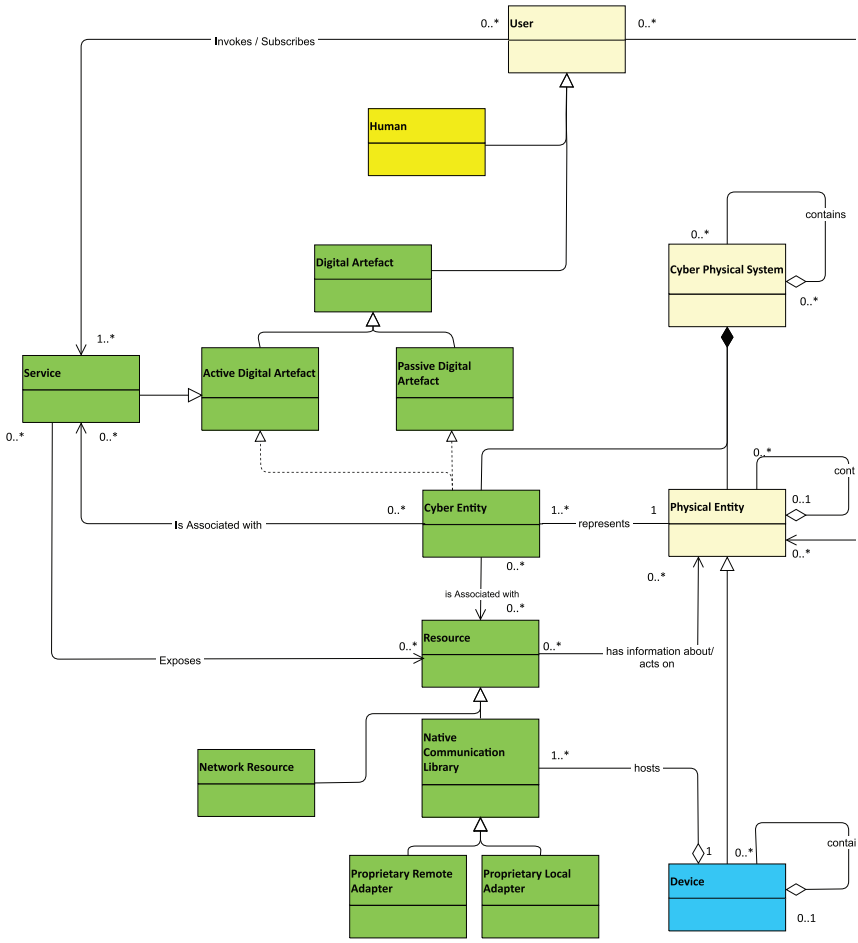

Figure 3.10 Domain model for MANTIS CPS.

issues are different in number, type and location if a cloud-based or an edge-based pattern is used and/or a facade, broker or mediator [Martin, 2002] pattern is applied. While the CPS level issues are related to the design and development of a given CPS, i.e., to provide guidance and guidelines on how to virtualize physical entities (i.e., machines) in terms of services/functionalities, especially for those that are low-tech. This is connected to the information level that is related to the description of the data and the messages/structures exchanged, processed and stored. The messages/structures exchanged here are connected to a MIMOSA-compliant

database that models the specific application context by using the MIMOSA OSA-EAI standard. Finally,interaction modeling is related to the definition and the identification of the necessary MEPs, i.e., how messages/structures are exchanged within the MANTIS platform.

The semantic data representation and exchange (see Figure 3.11) is aimed to describe the structure of all the data and/or information handled by cyber entities at a network level. Thus, the provided model details the way information should be modeled inside the cyber entity of a CPS and represents a necessary condition to guarantee that all the data circulating within the MANTIS platform cyberspace satisfies a well-defined structure to assure interoperability between different digital artifacts.

In here, the user can be a human person or some kind of a Digital Artefact (e.g., a Service, an application, or a software agent) that needs to interact with a Physical Entity, where a digital artifact is a software component.

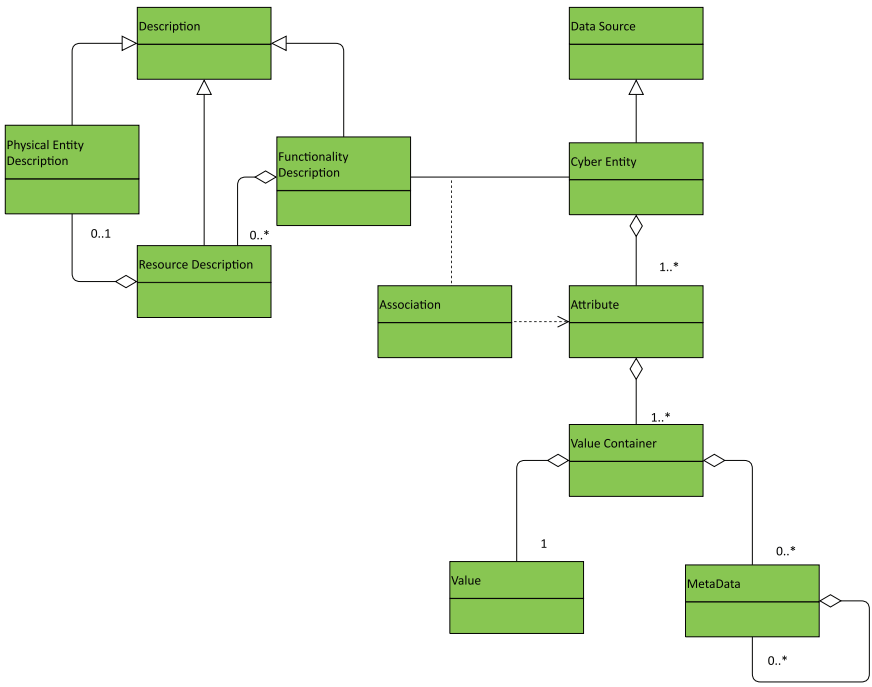

Figure 3.11 Semantic Data representation and information exchange model.

A Cyber Entity is represented in the digital world of Physical Entities. The Cyber Entities have two fundamental properties: (i) they are digital artifacts and (ii) they are the synchronized representation of a given set of aspects of their Physical Entities. Any change in the Physical Entity affects the Cyber Entity and vice-versa.

A resource is a software component that provides data from or is used in the actuation of Physical Entities. Since it is the Functionality that makes a Resource accessible, the relations between Resources and Cyber Entities are modeled as associations between Cyber Entities and Functionality. Resources can run on Devices or somewhere in the network (Network Resources). Devices host the technological interface (Native Communication Library) for interacting with, or gaining information about the Physical Entity.

The obvious similarities between IoT and CPS-based systems are always pushing the merging of the two research streams. The main aspects are represented by the elements *Cyber Entity, Functionality Description and Association*. A *Cyber Entity* is the cyber counterpart of a *Physical Entity* and the *Functionality Description* describes the set of functionalities the cyber entities are sharing within the virtual space. Actually, a functionality can be mapped to a service if the SOA technology [Thomas, 2008] is used or a skill/capability in a MAS [Jacques, 1999]. The Association is used to establish the connection between the *Attribute* of a cyber entity and the *Functionality Description*. As an example, for a temperature sensor a functionality could be the getTemperature function that provides information about the temperature *Attribute* value. A cyber entity can have zero to many different attributes.

Each *Attribute* has a name (attributeName), a type (attributeType), and one to many values (*Value Containers*). The attributeType specifies the semantic type of an attribute, for example, that the value represents temperature. Each value container groups one *Value* and zero to many *Metadata* fields that belongs to the given value. The metadata can, for instance, be used to save the timestamp of the value, or other quality parameters, such as accuracy or the unit of measurement. The cyber entity is also connected to the functionality description via the *Functionality Description – Cyber Entity* association. Additionally, it may contain one (or more) *Resource Description(s)*. Finally, the resource description might contain information about the physical entity. The concept of Value within the MANTIS information model is specified according to the OSA-CBM open standard (see Figure 3.12).

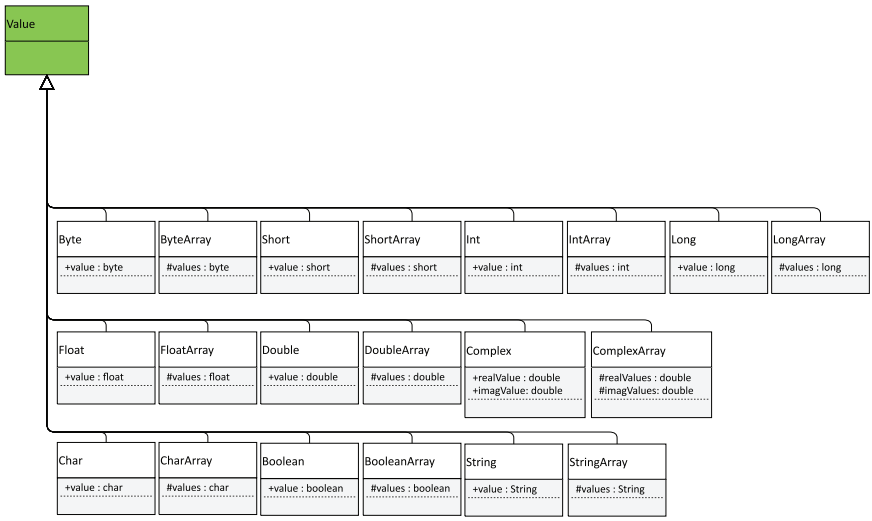

Figure 3.12 Value concept specification.

3.4.2.2 System interaction model

Significant actions, incidents or episodes need to be registered and stored in the MANTIS platform that promotes monitoring or data analysis for performance improvement. Those events store data relevant to the entities related in the process (e.g., temperature of a surface) but additionally must collect both spatial and temporal information and associate them to entities/measures (e.g., Cyber Entity controlling rolling sheets or temperature). Events might be triggered and/or should be created after a given situation (such as after sensor reading, an operational action, a breakdown or other maintenance actions).

3.4.2.2.1 *MANTIS event model*

Events are a fundamental part within MANTIS for supporting system interactions (at cyber level) at both edge and platform levels while interlinking data and automatic machine data processing. Consequently, a generic event model (see Figure 3.13 has been designed to provide the skeleton for the definition of all the events produced/consumed within the MANTIS platform.

The base type of all events is the *Abstract CEP Event* type. It provides the skeleton and basic information for modelling all the events within the MANTIS platform. This basic information is the *DateTime*,

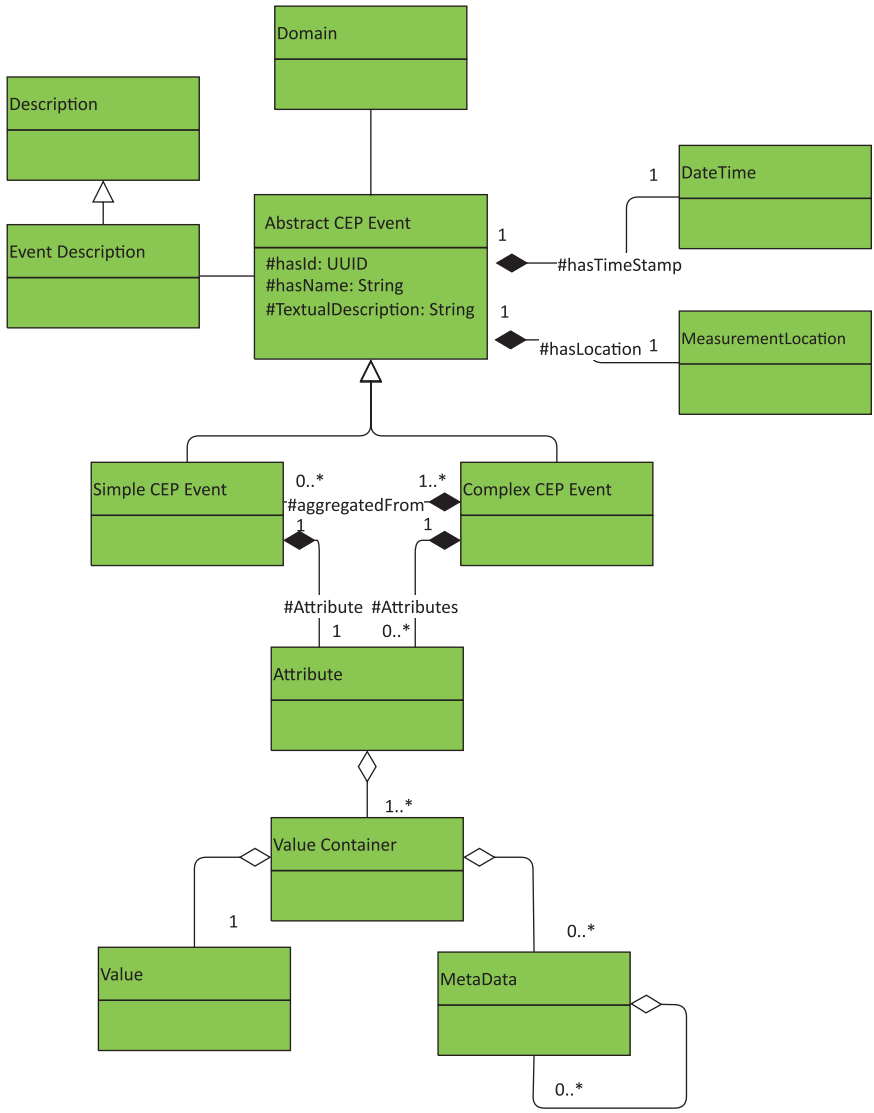

Figure 3.13 MANTIS event model (based on the IoT-A event information model).

the *EventDescription* and the *MeasurementLocation*. The concepts/classes DateTime and *MeasurementLocation* are defined to adhere to the OSA-CBM standard (the same is for the unique identifier type UUID).

There are two very generic concrete events type: i) the *Simple CEP Event*, that contains atomic event information and ii) the *Complex CEP Event* that

contains information derived by a complex event processing application. The *Value Container*, *Value* and *MetaData* concepts/classes are used to model the content of each event that in turn can be as simple as a temperature value to complex strings that are serialized objects.

3.4.2.2.2 *Patterns for interactions*

The definition of the most suited MEPs in the context of distributed computed is a typical problem [Martin, 2002]. MEPs refers to the way messages are exchanged between distributed components. It is worth noting that the selection and usage of a message exchange pattern can affect the way digital artifact should be implemented in order to be interoperable. In the context of MANTIS, three type of MEPs are considered, namely:

- Push/Fire-and-forget: messages are sent in between digital artefacts and with CPSs. The sender sends the message and the receiver receives the message and ends the message exchange activity;
- Request/Response-Reply: request response messages are sent in between digital artefacts and with CPSs. The sender sends a request message and the receiver receives the message and informs the sender with a response;
- Publish/Subscribe: messages are sent in between digital artefacts and with CPSs in the form of events. The event source publishes a topic and all the digital artefacts and/or CPS that are interested to the topic subscribe to it and will receive events.

The types of CPS and/or more in general digital artefacts that can be found within the MANTIS platform are: MANTIS-enabled that are supposed to be natively MANTIS platform compatible. These CPSs and/or digital artefacts already support the MANTIS interoperability specifications and can be immediately integrated within the MANTIS platform. These digital artefacts and/or resources need mechanisms and/or additional work in order to be integrated within the MANTIS platform. In this case, external adapters are needed to harmonize and bring together them within the MANTIS platform, i.e., translations from native to common protocols used in MANTIS.

3.4.2.3 Implementation integration

The model for implementation integration (see Figure 3.14) has been developed to capture and show how the provided interoperability models can be related to a concrete MANTIS platform instantiation with specifying and/or establishing technologies used.

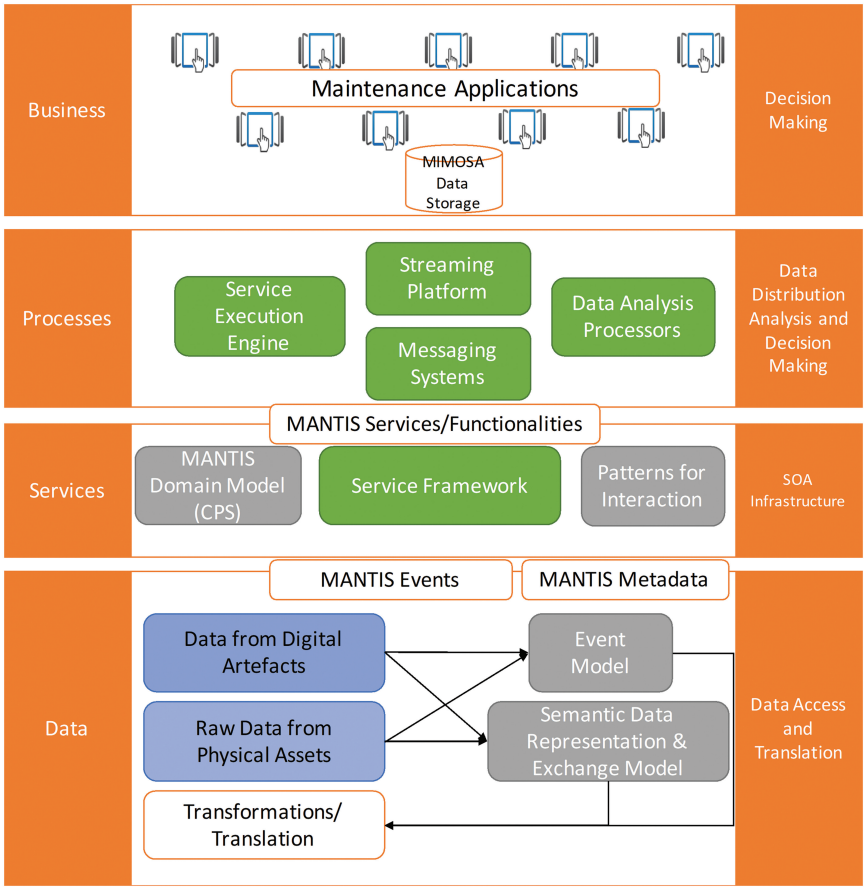

Figure 3.14 Model for technical integration.

The business model describes the specific domain for the software solution that needs to be implemented. The software model represents the instantiation of the semantic data representation and exchange model and system interaction model in the specific domain derived from the business model. Therefore, in the software model all the necessary aspects of the conceptual integration are included. Finally, the technical architecture represents and describes the concrete environment, infrastructure and related technologies for supporting the platform/application.

The technical framework of the MANTIS platform can be distilled into a model (see Figure 3.14) focused on the interoperability perspective. The model is represented by a 4-tier model that covers all the necessary issues and

aspects that developers need to consider whenever they want to implement MANTIS-compliant systems, i.e., data representation and exchange, system interactions (event models and patterns for interaction), data transformation and translation and services and functionalities definition, as well as, physical entities virtualization (domain model). A central part of the framework is the MIMOSA data storage that acts as a facilitator for the design and implementation of maintenance applications within the business tier.

3.5 Information Security Model

Traditionally, security has been just a commodity but along the years this perception has being changing to become an integral and an inseparable part of any system. Indeed, security nowadays is a functional requirement to become interoperable with many existing systems. In this sense, MANTIS addresses these requirements and has been not only focusing in providing functionalities but being secure by design. This aim can only be accomplished by means of a modern secure information model and the most suitable access control information system. In order to achieve this goal is necessary to understand the basic pillars of the information security [Rahalkar, 2016]:

- *Integrity*: to maintain the completeness and accuracy of data over its entire lifecycle;
- *Confidentiality*: to guarantee the privacy of data over its entire lifecycle to unauthorized individuals, entities or processes;
- *Availability*: to guarantee that the information is available when is needed.

Nevertheless, these principles impose various requirements towards the architecture. The system must be defined with having

- its every relevant element supplied with a digital identity;
- a specific information model for managing different levels of confidentiality;
- that is enforced by a security policy model.

From these requirements the right process for obtaining a realistic and effective security management system involves the following processes:

- Process 1. Digitization or the process of obtaining a unique and distinguishable digital identity;
- Process 2. Definition of an information model;

- Process 3. Definition of a control access policy specification;
- Process 4. Definition of additional requirements associated with MANTIS.

Finally, practical consideration is detailed in order to fulfill with existing security technologies such as advanced threat detection techniques.

3.5.1 Digital Identity

In MANTIS every object, subject and action must have a digital reflect recorded. Therefore, it is necessary to establish a specification and classification of the elements of the system, that can be involved in the processing of the information. In this context, digital identities are the key to be able to establish effective and realistic security policies, without them it is not possible to control the behavior of the system. Having in mind, MANTIS associates every element of the platform with one of the following categories:

- *Subject* is the element in charge of requesting operations (actions) with objects. These are the actors of the system. In many situations the subjects are processes intermediated by users but there are other situations where the processes are not associated with users;
- *Action* is the definition of an operation; every operation must be defined in order to control the behavior of the system;
- *Object* are the elements, which receive the actions. In this category, certain elements can be subjects and/or objects such as processes.

Nowadays, the process of giving an identity to an object/subject is performed by generating a digital certificate, and an unique identifier [Vacca, 2004], as depicted in Figure 3.15.

Figure 3.15 Elements having their digital identity (certificate).

3.5.2 Information Model

In previous steps, the digital identity for every element of MANTIS was defined. Now, it is necessary to define the security classifications of information while taking into account the potential of the platform under two challenging situations:

- Industrial environments. The companies try to maintain the availability of the system as well as the confidentiality of the information of the processes (key performance indication, KPI), the model of machines, the technology used, etc. In addition to this, there are two factors that become very important:
  - The integrity of the industrial processes is very important for taking right decisions on processes;
  - The system must respect operational safety systems. MANTIS cannot interfere with real-time systems, since that may produce personal injuries or catastrophic losses.
- Medical environments. The personal health information is one of the most critical assets, very restrictive legislation exist a in many countries related to that (c.f. GDPR [Donnelly, 2015]). MANTIS should implement a model, which guarantees this confidentiality of the information.

The assessment of these two challenging environments leads the use of restrictive security information models. A priori, the first candidate is the most restrictive information model known as Bell LaPadula (BLP) [Hansche et al., 2003]. This model is used in government and military applications and it is focused in enforcing the access control to confidential data. This model has the following properties, see Figure 3.16:

- The simple security property. This establishes that a subject of a specific level cannot read information at a higher security level;
- The *(star) property. This establishes that a subject of a specific level cannot write to any object at a lower security level;
- The discretionary security property. In this the specification of the discretionary access control is made by means an access matrix;
- Security levels introduced: Top Secret, Secret, Confidential and Unclassified.

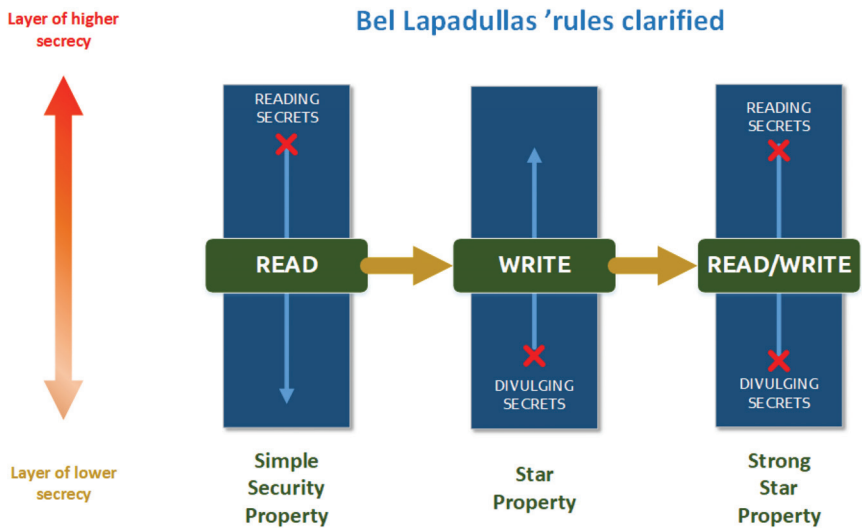

Figure 3.16 Elements of the BLP model [Hansche et al., 2003].

3.5.3 Control Access Policy Specification

The information model requires an access control policy to ensure security of the system. With this aim in mind, the original model specified (BLP) establishes the policies as a matrix where the access to every element is specified. The main drawback of the original specification here is related to the inherent complexity of MANTIS, which requires a more complex specification of security, and at the same time, a way to facilitate the management of the security. The direct application of this matrix will conclude in a huge matrix and poses significant problems to manage the policy in real life. With the aim of overcoming this drawback, an initial approach is trying to facilitate with the use of the concept of Roles. It is a mechanism for grouping sets of subjects with the same level of security but with some other interesting properties in terms of manageability:

- Encapsulating the organizational functions/duties of a user;
- Different roles can be defined, each for different types of competences, which are then assigned to users;
- Realizing the security principle of "least privilege" [Rahalkar, 2016];
- It is consistent with BLP model.

Therefore, in MANTIS, the first approach to manage the security will be the use of roles and BLP for establishing the security policy. The NIST

establishes some subdivisions of the original model such as: Core RBAC, Hierarchical RBAC, Constraint RBAC and Consolidated Model [Ferraiolo and Kuhn, 1992]. An RBAC model can formally be described by the tuple RBAC=$< U, R, P, O, >$, the most important elements of this tuple are:

- U: User;
- R: Role;
- P: Permission;
- O: Object.

The model that best suits to MANTIS is the *Hierarchical and Constrained RBAC* model [Ferraiolo and Kuhn, 1992], which supports challenging environments and situations. The joint use with the BLP model facilitates the management of the security of the system. MANTIS uses the standard for specifying the security policy called eXtensible Access Control Markup Language (XACML) [OASIS, 2018]. This standard provides

- a Policy Language;
- a Request and Response Language;
- Standard data-types, functions, combining algorithms.

It is extensible, where there can be privacy profiles, with architecture defining the major components in an implementation. The structure of a security policy is specified in Figure 3.17 and the relevant, general terms within XACML are the following:

- Resource: Data, system component or service;
- Subject: An actor who requests to access certain Resources;
- Action: An action on resource;
- Environment: The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action;
- Attributes: Characteristics of a subject, resource, action or environment;
- Target: Defines conditions that determine whether policy applies to request.

3.5.4 Additional Requirements

MANTIS is an agnostic reference architecture, which can be applied in many environments, but like many technologies it is necessary to have high adaptibility. In this sense, many situations might exist where the classic security information model and their corresponding access control policies

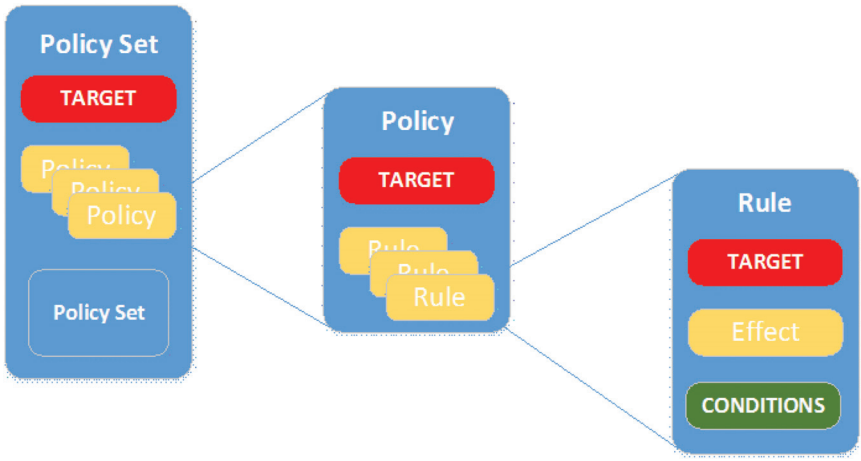

Figure 3.17 Policy hierarchy.

are instantiated; however there might be other situations involve relevant additional changes (e.g., the integration of new sensors in every machine that the company sells). This innocent change might cause a nightmare in terms of efficiency when only classic security access controls are in place. Taking these aspects in MANTIS into account, it will be necessary to consider the nature of changes to be introduced for improving the efficiency of the security management.

MANTIS considers not only the joining of the RBAC [Ferraiolo and Kuhn, 1992] and BLP [Hansche et al., 2003], but the use of modern approaches, that is, an ABAC model. The solution relies on the use of additional PKI [Vacca, 2004] in a different way to support the sharing of responsibilities. With these, emphasis is put on having a controlled but shared access model. The access to information is conditioned by the security policy defined in the cloud, managed by the company. This allows controlling situations such as validity periods for accessing the information but also it would be beneficial to the original company to be able to control external parties interesting in their data. This leads to work with a federated PKI for sharing Certification Authorities and to introduce the concept of secret sharing schemes. An increase in the number of sensors also requires additional considerations. In order to alleviate this, MANTIS introduces the concept of attributes in final elements for controlling the access by using existing RBAC based policies.

3.6 Architecture Evaluation

Software and systems design, in its core essence, is the creative activity of software engineers making principal design decisions about a software system to be built or to be evolved. It translates the concerns and drivers in the problem space into design decisions and solution concepts in the solution space. Architecture evaluation is a valuable, useful, and a worthwhile instrument to manage risks in software engineering. It provides confidence for decision-making at any time in the lifecycle of a software system [Knodel and Naab, 2016].

3.6.1 Architecture Evaluation Goals, Benefits and Activities

Architecture evaluation is the key quality-engineering instrument in software and systems design. Its goal is to make sure that the resulting systems really exhibit the desired qualities. To this end, it pursues two major objectives:

On the one hand, architecture evaluation aims at improving the overall quality of software and systems design. Architecture evaluations challenge the decisions made. They help clarifying quality requirements and enable to analyse the adequacy of the architecture solution. Further, they allow predictions of the impact of the architecture solutions and consequently the decisions made on the quality of the resulting system. As the architecture sets the course for the resulting system, quality problems and drawbacks of decisions can be detected early. Thus, architecture evaluation serves to mitigate risks. Eventually, it enables the improvement of the architecture by correcting and adapting the decisions made. Furthermore, architecture evaluation comes along with the side effect of increased architecture awareness in the development organization. While reasoning about and communicating the decisions made, their understanding in the organization is improved. Architecture evaluation reveals the rationales for the decisions, and their justification allows achieving a common understanding in the development organization. In short, architecture evaluation determines how well suited the architecture of the system for its purpose is.

On the other hand – once having a well-designed architecture – architecture evaluation aims as well at preserving architectural decisions and quality in the evolution of software systems. The follow-up activities in the lifecycle of a software system first translate architectural decisions into component models, detailed design models, and eventually source code including data structures and algorithms. At later points in time, evolving requirements and change requests yield modified system artefacts. However, to reap the architectural investment benefits, the managed software system

lifecycle needs to enforce and preserve the architectural decisions made. This sustainment of architectural decisions assures that the architecture is in fact the conceptual tool to cope with challenges in its evolution. To be able to evaluate the decision enforcement, architecture documentation must trace all decisions to the system artefacts. Traceability of architectural decisions breaks down into the accuracy of their description and the distance of the system to its architecture.

With this said, the benefits of architecture evaluation turn out to be an improved software architecture, improved architecture documentation, and improved implementations of architectural solutions. The evaluation activities help to mitigate risks by raising the likelihood to detect problems early and to clarify the required system qualities. Evaluation also improves the understanding of design decisions and leads to a higher awareness of the architecture in organizations. As needed for input, the traces from architectural decisions to the system artefacts help their preservation and allow for higher compliance in implementations.

To achieve the above-mentioned objectives, MANTIS follows the RATE approach [Knodel and Naab, 2016], developed by Fraunhofer. RATE is a compilation and collection of best practices of existing evaluation approaches tailored towards a pragmatic (or rapid) application in industry. It comprises five checks, whereby each check serves for a distinct purpose. All checks follow the same working principle: to reveal findings to confirm and improve the system quality and/or the artifact quality.

3.6.2 Concepts and Definitions

RATE uses the following concepts and definitions in its analysis:

Stakeholder: A stakeholder in a software architecture is a person, group, or entity, with an interest in or concerns about the realization of the architecture [ISO/IEC/IEEE, 2011].

Concern: A concern about an architecture is a requirement, an objective, an intention, or an aspiration that the stakeholder has for the architecture [ISO/IEC/IEEE, 2011].

Usually the stakeholder concerns are not consolidated or validated. This makes dealing with the stakeholder concerns challenging, since concerns might be ambiguous, conflicting with other stakeholders' concerns, and are likely to be incomplete. In case a concern is specific, unambiguous, and measurable, it is possible to call it a *requirement* concern; otherwise, architecture scenarios can capture concerns. In either case, stakeholder

concerns form the product and drive the architecture, which explains the need for compensation of missing and too complex concerns, for their aggregation and consolidation as well as their negotiation in case of ambiguity and inconsistency. This the architect does by deriving architecture drivers from the concerns.

Architecture Driver: In general, it can be drawn a distinction between four main classes of architecture drivers: business goals, functional requirements, constraints, and quality requirements. Each of these classes might have its individual stakeholders that articulate concerns belonging to that particular class. In other words, all drivers originate from stakeholders in one way or another. The identification and analysis of stakeholders for further requirements elicitation and their stake within the architecture development is therefore key to any architecture definition or evaluation.

Business Goals: are the first class (and most abstract) of architectural drivers. Business goals are goals that are important for the overall enterprise that is developing the respective architecture or has placed an order to build the system. Usually the business goals are quite abstract and are only partially depending on the architecture under consideration. However, the business goals are the most essential ones, since without the business goals there would be no need to think about creating an architecture of (a set of) products that end up in supporting a business goal. Examples for business goals are time to market (denoting the strategy in terms of time), the market scope, or costs.

Functional Requirements: are drivers for the architecture as well. However, there are differences in functional requirements: some drive the architecture – some do not. It depends on characteristics like "Does this particular function separate us from competitors' products?" In some sense, the functional requirements that make the product unique and worth building are the ones that influence the architecture development the most. These kinds of functional requirements the architecture needs to explicitly support; otherwise, the endeavour of building an architecture would be meaningless.

Quality Requirements: Quality is not only about correctness of functionality. Successful software systems have to assure additional properties such as performance, security, extensibility, maintainability, and so forth. In general, it is possible to distinguish between run-time and development-time quality attributes. Run-time quality attributes can be measured by watching the respective system in operation. Examples for run-time quality attributes are performance, security, safety, availability, and reliability. Development-time

quality attributes can be measured by watching a team in operation. Examples for development-time quality attributes are extensibility, modifiability, and portability. One problem, however, is that there is no standard measurable meaning of quality attributes. Besides naming issues, people also tend to create new notions on their own. The solution towards this problem is to utilize the so-called architectural scenarios (see *Architecture Scenario*) that make the meaning of the quality attribute in the system context clearer and that lower the chance for misinterpretations.

Constraints: One important but easily overlooked input for software and systems design are constraints that influence the design decisions of subsequent steps. Constraints can be organizational, technical, regulatory, or political. Organizational constraints might arise from the resources available for a particular system development effort. Technical constraints might arise from legacy systems that are already deployed in the field. Regulatory constraints usually stem from obligations to comply with particular standards. Depending on the domain, there might be different standards to consider. Political constraints are most of the time disguised as technical constraints. There might be different (more or less reasonable) roots for the existence of the constraint, however, since the source of the constraint is most likely higher management there is only low negotiability of the constraint from the perspective of the architects. Making them explicit, however, provides a solid basis for subsequent decision making in design.

Architecture Scenario: An architectural scenario is a crisp, concise description of a situation that the system is likely to face, along with a definition of the response required of the system [Rozanski and Woods, 2011]. Architecture scenarios could be used to document both, functional and quality drivers in a measurable way, but are especially used for capturing software and system qualities. Functional requirements are usually clearer than quality requirements. However, it is the quality requirements that drive the architecture most. Therefore, it is crucial to elicit required qualities using scenarios in a measurable way, so that architects or evaluators can find a baseline to work with. The scenarios are the input for creating, designing and evolving architectural solutions, which have to be preserved in follow-up activities. Thereby scenarios evolve over distinct states: Unknown, Elicited, Designed, Documented, Implemented, and Sustained. Depending on the state of the scenario, different types of architecture evaluations are possible. Architecture scenarios should be documented in a structured way (cf. [Clements et al., 2010]), rendering data on the following aspects:

- Scenario: Representative name (and ID) of the scenario;
- Quality: Related quality attribute;
- Environment: Context applying to this scenario (if possible provide quantifications);
- Stimulus: The event or condition arising from this scenario (if possible provide quantifications);
- Response: The expected reaction of the system to the scenario event;
- Response Measure: The measurable effects showing if the scenario is fulfilled by the architecture.

3.6.3 Architecture Evaluation Types

RATE (Figure 3.18) comprises five checks on three evaluation levels, whereby each check focuses on different aspects. The main levels are stakeholder level, architecture level and implementation level. In the stakeholder level, architecture drivers are validated, in the architecture level, the architecture and its documentation are checked and in the implementation level, the compliance and code quality is checked.

All checks performed come with a related rating of the confidence level of the findings.

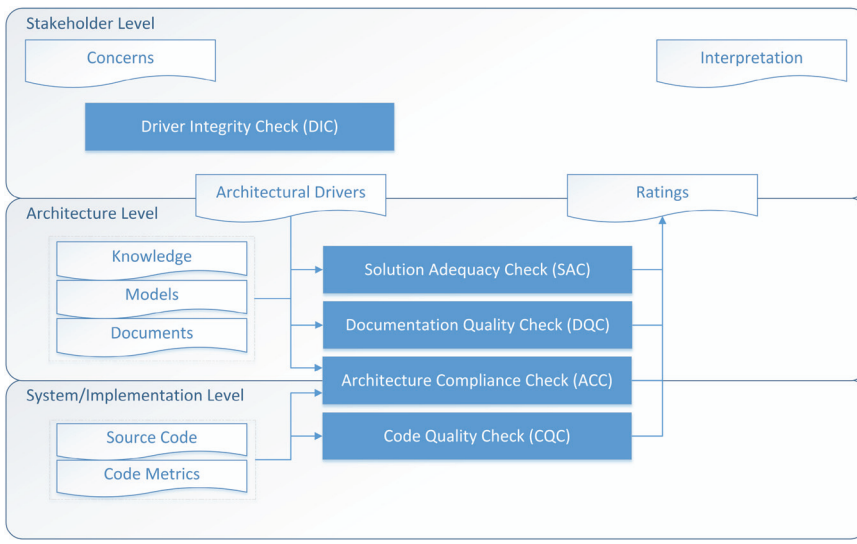

Figure 3.18 Architecture evaluation with Fraunhofer RATE [Knodel and Naab, 2016].

The rating comprises

- the severity of findings that expresses the criticality of the findings aggregated over all findings per goal;
- the balance of findings that expresses the ratio of positive vs. negative findings aggregated per goal.

The combination of the ratings results in a rating of N/A (Not Applicable), or an assignment to one of the target achievement levels NO, PARTIAL, LARGE, and FULL where each check defines its own target achievements.

Driver Integrity Check (DIC): The goal of the Driver Integrity Check is to get confidence that an architecture is built based on a set of architecture drivers that is agreed among stakeholders and to clarify unclear or not agreed architecture drivers. DIC also aims to compensate not elicited requirements and to aggregate a large set of requirements into a manageable set for an architecture evaluation. Inputs of the DIC are stakeholder information (if available), existing documentation (if available), and a template for documenting architecture drivers (mandatory). It is crucial to elicit required qualities using scenarios in a measurable way, so that architects or evaluators can find a baseline to work with. One can use architecture scenarios to document both, functional and quality drivers in a measurable way.

Solution Adequacy Check (SAC): The Solution Adequacy Check can be done as soon as there is a first idea of the architecture, and tries to answer the question how well suited the architecture for the intended purpose is. The SAC

- can determine whether an architecture permits or precludes the achievement of targeted functional and quality requirements;
- can determine whether an architecture or part of it is concrete enough or redundant for the purpose it will be used for;
- enables the identification of problematic design decisions;
- enables the early prediction;
- enables a timely reaction and correction.

That is, the main purpose of the SAC is to gain confidence, to predict the future behaviour of the system or to get some evidences. Inputs for the SAC are architecture drivers and architecture documentation. Besides findings on the adequacy of architecture decisions to fulfil architecture drivers (explicit rationales, risks, trade-offs, assumptions) SAC puts out revised architecture decisions, driver solutions, and diagrams.

Documentation Quality Check (DQC): The Documentation Quality Check can be done as soon as there is a first draft of the architecture description, and tries to answer the question how well documented the architecture solution for its audience and purposes is. The DQC

- can determine whether or not an architecture documentation allows understanding the solution;
- enables information sharing on architecture;
- enables consistency checks;
- enables the detection of gaps in architecture and its documentation.

The main purpose of the DQC is to ease information sharing within architectural stakeholders. Inputs for the DQC are the documentation purposes, architecture documents, models, wikis, sketches, API documentation, and the targeted audience. It feeds back findings on adequacy of the documentation and its adherence to best practices.

Architecture Compliance Check (ACC): The Architecture Compliance Check can be done as soon as there is a first skeleton of the implementation, however, it should be iterated, as there may be modified implementations during the lifecycle of the system. The ACC tries to answer the question how well realized the architectural solution by the implementation is. The ACC

- can determine whether or not the implementation violates architectural solutions;
- enables traceability;
- enables compliance checking (as-is vs. planned intention);
- enables delta tracking (as-is vs. envisioned target).

The main purposes for ACC are to identify the structural and behavioural compliances between the architecture and the implemented system. Inputs for the ACC are architecture documents, models, wikis, sketches, API documentation, source code, and the running system (if applicable). It results in findings on the compliance of the implementation with respect to the intended architecture, convergences, divergences (violations) and absences (violations).

Code Quality Check (CQC): The Code Quality Check's main goal is to gather data about the source code base. As such, the CQC is not a direct part of architecture evaluation. However, reasoning about quality attributes

(in particular maintainability) requires the CQC results in order to make valid statements about the software system under evaluation. CQC helps to

- improve the implementation of a software system;
- monitor (and fix) anomalies over time;
- derive common metrics and coding best practices;
- define team-specific coding guidelines;
- improve the overall understanding of the code base;
- make the development organization more robust.

Inputs for the CQC are the source code, and build scripts (if applicable). It puts out findings on quality of the source code, best practice violations, code clones, quality warnings (maintainability, security), code metrics, and more.

3.7 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 chapter presents a generic platform together with specific toolset to cover the problem space. Typical target areas of proactive maintenance include FP, calculation of the RUL, 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.). There are various issues to tackle on each of the tiers, and even within the communications between the various actors and tiers. Therefore, the interoperability and security aspects must be focal points when such an installation is made, and hence their place in this Chapter.

Moreover, such an architecture has to be evaluated and has to facilitate the requirements of every actor and stakeholder. To this end, evaluation techniques are instruments to increase the confidence level in the architecture solutions, where the confidence level expresses trust in the architecture designed and the system derived from it. In addition, architecture evaluation allows the prediction of the impact of decisions made and helps secure the consistency across decisions in descriptions. It helps to check the structural and behavioral compliance of the implemented as-is architecture with the intended one, and makes possible the assertion of qualities in the system at execution. Therefore, architecture evaluation is the key quality-engineering instrument in software and systems design, and it provides confidence for decision-making at any time in the life cycle of a software system.

This architecture already has been successfully utilized in various use-cases from industrial utility vehicles (forklifts) through railway control system to CPPSs [Ferreira et al., 2017; Hegedüs et al., 2018], as part of the ECSEL MANTIS project [The MANTIS Consortium, 2018]. These use case instantiations are presented in Chapter 7.

References

Albano, M., Barbosa, P. M., Silva, J., Duarte, R., Ferreira, L. L., and Delsing, J. (2017). Quality of service on the Arrowhead Framework. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS),* (pp. 1–8). IEEE.

Amazon. (2017) Amazon web services. https://aws.amazon.com/products/analytics/.

Apache. (2017) Cassandra database. http://cassandra.apache.org/.

Apache. (2017) Hadoop distributed file system. http://hadoop.apache.org/.

Apache. (2017) Spark. https://spark.apache.org/.

Apache Community'. (2017) The kafka distributed streaming platform. http://kafka.apache.org/.

Apache Community. (2017) Storm stream processor. http://storm.apache.org/.

Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A. S., and Buyya, R. (2015) 'Big data computing and clouds: Trends and future directions', *Journal of Parallel and Distributed Computing*, 79, pp. 3–15.

Bell, M. (2008) *Introduction to Service-Oriented Modeling*. Wiley and Sons.

Bermbach, D., Wittern, E., and Tai, S. (2017) *Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective*. Springer.

Cengarle, M., Bensalen, S., McDermid, J., Passerone, R., Sangiovanni-Vincetelli, A., and Torngren, M. (2013) *Characteristics, Capabilities, Potential Applications of Cyber-physical Systems: A Preliminary Analysis*.

Cisco. (2016) Global cloud index: Forecast and methodology. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

Clements, P., et al. (2010) *Documenting Software Architectures: Views and Beyond*. p. 608.

Curry, E. (2004) 'Message-oriented middleware,' *Middleware for Communications*, pp. 1–26.

Delsing, J. (2017) *IoT Automation: Arrowhead Framework*, CRC Press.

Di Orio, G., Maló, P., Barata, J., Albano, M., and Ferreira, L. L. (2018, July). Towards a Framework for Interoperable and Interconnected CPS-populated Systems for Proactive Maintenance. *In 2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, (pp. 146–151). IEEE.

Donnelly, C. (2015) Eu data protection regulation: What the ec legislation means for cloud providers. http://www.computerweekly.com/.

Fan, W. and Bifet, A. (2013) 'Mining big data: Current status, and forecast to the future,' *ACM sIGKDD Explorations Newsletter*, 14(2), pp. 1–5.

Ferraiolo, D. F. and Kuhn, D. R. (1992) 'Role-based access control,' In *15th National Computer Security Conference*, Baltimore, October 13–16, pp. 554–563.

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

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2015) *Systems and Methods for Replicating Data*. US Patent 9,047,307.

Grover, P. and Kar, A. K. (2017) 'Big data analytics: A review on theoretical contributions and tools used in literature', *Global Journal of Flexible Systems Management*, pp. 1–27.

Han, J., Haihong, E., Le, G., and Du, J. (2011) 'Survey on nosql database,' In *2011 6th International Conference on Pervasive Computing and Applications (ICPCA)*, pp. 363–366.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. (2015) 'The rise of big data on cloud computing: Review and open research issues,' *Information Systems*, 47, pp. 98–115.

Hansche, S., Berti, J., and Hare, C. (2003) *Official (ISC)2 Guide to the CISSP Exam*. CRC Press. p. 104. ISBN 978-0-8493-1707-1.

Hausenblas, M. and Bijnens, N. (2017) 'The lambda architecture website,' http://lambda-architecture.net/.

Hecht, R. and Jablonski, S. (2011) 'Nosql evaluation: A use case oriented survey,' In *2011 International Conference on Cloud and Service Computing (CSC)*, pp. 336–341.

Hegedüs, C., Ciancarini, P., Franko, A., Kancilija, A., Moldovan, I., Papa, G., Poklukar, S., Riccardi, M., Sillitti, A., and Varga, P. (2018) 'Proactive maintenance of railway switches,' In *Proceedings of the 5th International Conference on Control, Decision and Information Technology (CoDIT)*, Thessaloniki, Greece.

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.

Industrial Internet Consortium. (2017) The industrial internet of things reference architecture.

International Electrotechnical Commission. (1993) *Information Technology – Vocabulary – Part 1: Fundamental terms*.

International Electrotechnical Commission. (2003–2007). *Enterprise-control System Integration*.

IoT-A Reference Architecture Model. (2018) http://open-platforms.eu/standard_protocol/iot-a-architectural-reference-model/.

ISO. (2012) Condition monitoring and diagnostics of machines – data processing, communication and presentation.

ISO/IEC/IEEE. (2011) 'Systems and software engineering – architecture description,' *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46.

Jacques, F. (1999) *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Jantunen, E., Di Orio, G., Hegedus, C., Varga, P., Moldovan, I., Larrinaga, F., Becker, M., Albano, M., and Malo, P. (2018) Maintenance 4.0 world of integrated information.

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

Kappa. (2018) 'The kappa architecture site,' https://www.talend.com/blog/2017/08/28/lambda-kappa-real-time-big-data-architectures/.

Knodel, J. and Naab, M. (2016) *Pragmatic Evaluation of Software Architectures*, p. 132.

Laney, D. (2001) '3d data management: Controlling data volume, velocity and variety,' *META Group Research Note*, 6, p. 70.

Lebold, M. and Thurston, M. (2001) 'Open standards for condition-based maintenance and prognostic system,' In *Proceedings of MARCON 2001 – Fifth annual maintenance and reliability conference*, Gatlinburg, USA.

Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. (2009) 'What's inside the cloud? An architectural map of the cloud landscape,' In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pp. 23–31. IEEE Computer Society.

Martin, R. C. (2002) *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall.

Martinez-Fernandez, S., Dos Santos, P. M., Ayala, C., Franch, X., and Travassos, G. (2015) 'Aggregating empirical evidence about the benefits and drawbacks of software reference architectures,' In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Beijing.

Microsoft. (2017) 'Azure web services,' https://azure.microsoft.com.

MIMOSA consortium. (2016) 'The mimosa project site,' http://www.mimosa.org/.

Mineraud, J., Mazhelis, O., Su, X., and Tarkoma, S. (2016) 'A gap analysis of internet-of-things platforms,' *Computer Communications*, 89, pp. 5–16.

Munshi, A. A. and Yasser, A.-R. I. M. (2017) 'Big data framework for analytics in smart grids,' *Electric Power Systems Research*, 151, pp. 369–380.

Noiumkar, P. and Chomsiri, T. (2014) 'A comparison the level of security on top 5 open source nosql databases,' In *The 9th International Conference on Information Technology and Applications (ICITA2014)*.

Nokia. (2017) 'Disco distributed file system,' https://disco.readthedocs.io.

OASIS. (2018) 'eXtensible Access Control Markup Language (XACML),' https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#CURRENT.

Pohl, K., Hönninger, H., Achatz, R., and Broy, M. (2012) *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer, 2012.

Rahalkar, S. A. (2016) *Information Security Basics. In Certified Ethical Hacker (CEH) Foundation Guide*, Apress, Berkeley, CA, pp. 85–95.

Rozanski, N. and Woods, E. (2011) *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, p. 678.

Sahafizadeh, E. and Nematbakhsh, M. A. (2015) 'A survey on security issues in big data and nosql,' *Advances in Computer Science: an International Journal*, 4(4), pp. 68–72.

Singh, R., Singh, K., et al. (2010) 'A descriptive classification of causes of data quality problems in data warehousing,' *International Journal of Computer Science Issues*, 7(3), pp. 41–50.

The MANTIS Consortium. (2018) 'The mantis project website,' http://www.mantis-project.eu/.

Thomas, E. R. L. (2008) *SOA: Principles of Service Design.* Upper Saddle River: Prentice Hall.

Vacca, J. R. (2004) *Public Key Infrastructure: Building Trusted Applications and Web Services*, CRC Press. ISBN 978-0-8493-0822-2.

Wapice. (2018) IoT-Ticket. https://iot-ticket.com/.