DSLs and Middleware Platforms in a Model-Driven Development Approach for Secure Predictive Maintenance Systems in Smart Factories

Jobish John[1,3(✉)], Amrita Ghosal[1,4(✉)], Tiziana Margaria[1,2,4(✉)], and Dirk Pesch[1,3(✉)]

[1] CONFIRM, SFI Research Centre for Smart Manufacturing, Limerick, Ireland
[2] Lero, SFI Research Centre for Software, Limerick, Ireland
[3] School of CS and IT, University College Cork, Cork, Ireland
{j.john,d.pesch}@cs.ucc.ie
[4] CSIS, University of Limerick, Limerick, Ireland
{amrita.ghosal,tiziana.margaria}@ul.ie

Abstract. In many industries, traditional automation systems (operating technology) such as PLCs are being replaced with modern, networked ICT-based systems as part of a drive towards the Industrial Internet of Things (IIoT). The intention behind this is to use more cost-effective, open platforms that also integrate better with an organisation's information technology (IT) systems. In order to deal with heterogeneity in these systems, middleware platforms such as EdgeX Foundry, IoTivity, FI-WARE for Internet of Things (IoT) systems are under development that provide integration and try to overcome interoperability issues between devices of different standards. In this paper, we consider the EdgeX Foundry IIoT middleware platform as a transformation engine between field devices and enterprise applications. We also consider security as a critical element in this and discuss how to prevent or mitigate the possibility of several security risks. Here we address secure data access control by introducing a declarative policy layer implementable using Ciphertext-Policy Attribute-Based Encryption (CP-ABE). Finally, we tackle the interoperability challenge at the application layer by connecting EdgeX with DIME, a model-driven/low-code application development platform that provides methods and techniques for systematic integration based on layered Domain-Specific Languages (DSL). Here, EdgeX services are accessed through a Native DSL, and the application logic is designed in the DIME Language DSL, lifting middleware development/configuration to a DSL abstraction level. Through the use of DSLs, this approach covers the integration space domain by domain, technology by technology, and is thus highly generalizable and reusable. We validate our approach with an example IIoT use case in smart manufacturing.

An earlier version of this work was presented at the Irish Manufacturing Council's International Manufacturing Conference (IMC 37) 2021, Athlone, Ireland.

T. Margaria and B. Steffen (Eds.): ISoLA 2021, LNCS 13036, pp. 146–161, 2021.
https://doi.org/10.1007/978-3-030-89159-6_10

Keywords: Predictive maintenance · EdgeX-Foundry · Secured access · Domain-specific languages

1 Introduction

The Industrial Internet of Things (IIoT) enables the transformation of traditional manufacturing systems into highly flexible, scalable and smart interconnected automation systems widely known as Industry 4.0 [1,5,23]. Decentralized decision making through real-time monitoring, using a large number of networked devices form the basis of IIoT [21]. To realize future smart factories, many field devices operating with both wired and wireless communication technologies need to coexist and interoperate seamlessly, which is one of the significant challenges for IIoT adopters.

Extracting meaningful insights from the large volume of data generated by devices poses another significant challenge. It is usually done at two levels, at the on-premise network edge (widely referred to as edge computing) and at the cloud level (cloud computing) [3,21]. With advancements in edge computing, decisions can be taken at the local edge, especially in delay-sensitive situations. Remote monitoring, diagnosis and maintenance of complex equipment/ machines is one of such applications that is widely used in smart manufacturing. Maintenance of assets in a factory is carried out with three different approaches: i) Reactive maintenance (e.g., repair the system once it breaks down), ii) preventive maintenance (e.g., regular checks), and iii) predictive maintenance (e.g., IIoT smart sensing-based solution that predicts and schedules the machine maintenance at the right time) [7]. Predictive maintenance schemes provide the best utilization of assets by minimizing their downtimes.

In this paper we study a predictive maintenance system (PreMS) from a system integration point of view, encompassing heterogeneous system elements involved in an architecturally sound solution that provides reliable interoperability. These elements include IIoT components, middleware, specific application logic needed to build the PreMS solution, and also security aspects. In doing this, we leverage as far as is possible a platform approach instead of bespoke programming or even bespoke code-based integration. A well-designed IIoT platform at the network edge can act as a seamless service-based transformation engine between field devices and enterprise applications. To deal with the field device heterogeneity in a flexible, scalable and secured fashion, industry consortia have been developing several IoT middleware platforms such as EdgeX Foundry [10], IoTivity [22], FI-WARE [22]. To overcome interoperability issues between field devices of different standards we consider the EdgeX Foundry platform as our IIoT middleware of choice at the network edge.

For the solution design and implementation, we adopt the approach proposed in [15], which shows the use of a flexible, low-code platform for enhanced interoperability with characteristics well suited for our PreMS application. While [15] shows its use in a building automation setting, this flexible, low-code platform for enhanced interoperability serves our purposes for building an IIoT platform here too. Our contributions are

– an architectural approach to edge computing based integration and deployment of middleware based model driven software applications for IIoT including EdgeX and DIME, a graphical application development methodology backed by formal models [6]
– a role based, Ciphertext-Policy Attribute-Based Encryption (CP-ABE) security policy development approach for security, privacy and compliance policies compatible with DIME.

Our previous work [15] showed that such a combination can yield a uniform paradigm and platform for the design of both applications and their properties.

The remainder of the paper is structured as follows. Section 2 provides an overview of the EdgeX Foundry middleware and system integration platform. Section 3 presents the PReMS use case application including device provisioning, data and alert handling and reusable application development using Domain Specific Languages (DSLs). Section 4 presents the security policy approach and Sect. 5 concludes the paper.

2 EdgeX Foundry as IIoT Middleware Platform

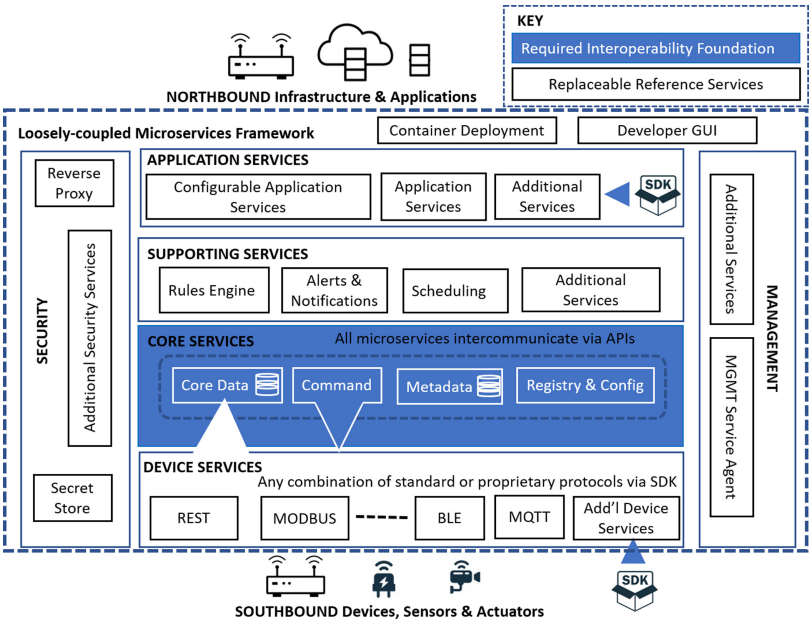

Fig. 1. EdgeX Foundry - platform architecture [10]

Although there are several IIoT middleware platforms available [22], we choose to use EdgeX Foundry at our network edge since it is an open-source, hardware agnostic, highly active, plug and play IIoT middleware platform aimed at edge computing. It is also supported by several industrial organisations, many

of which are partners in our national research centre [8]. A commercial-grade version of EdgeX and associated IIoT device connectors (detailed in Sect. 3.1) is also available [14]. EdgeX Foundry [10] consists of six different layers (four service layers + two system layers) as shown in Fig. 1, implemented using loosely coupled microservices that are container deployable. Details about each layer are discussed in [15]. External devices, such as sensors and actuators of different protocols interact with the EdgeX platform using device connectors present in the device service layer. EdgeX includes security elements for protecting data while managing IoT entities. As an open-source platform, security features are developed with open interfaces and pluggable, replaceable modules. The secret store is implemented through the open source Vault [15], and communication with microservices is secured using TLS. A secure API gateway is the sole entry point for EdgeX REST traffic and safeguards unauthorized access to EdgeX's REST APIs.

Recent literature shows how to exploit **EdgeX Foundry** as a ready-made middleware platform. Xu et al. [25] propose a microservice security agent that provides secure authentication by integrating the edge computing platform with an API gateway. Han et al. [13] designed a monitoring system using EdgeX to deal with diverse communication protocols and insufficient cloud computing resources. Kim et al. [17] design an EdgeX-based general-purpose, lightweight edge gateway with low-end CPU and low-capacity memory. The gateway processes small load data to monitor control systems for smart homes, smart farms, and smart meters. Zhang et al. [26] describe a trusted platform to preserve data privacy of edge computing clients via an edge secured environment that integrates EdgeX and the Hyperledger Fabric blockchain network. Platform portability is enhanced by using EdgeX and extending it to incorporate the Hyperledger via a collection of well-defined security authentication microservices. Xu et al. [24] present an EdgeX-based edge computing environment that covers implementation and deployment at the edge. Devices are connected via CoAP and HTTP-based REST APIs on a Raspberry Pi, showing experimentally that CoAP is more stable and faster than HTTP.

In [15] we showed how to design and develop a low-code application for building automation that uses EdgeX's capabilities as an integration service. For the specific low-code support, we used a model-driven approach based on Domain Specific Languages at two levels:

1. *language DSLs* as a mechanism to design and implement the application design environment itself, i.e. the Integrated Modeling Environment DIME, and
2. *application domain DSLs* at application design time. The *Native DSL* mechanism in DIME is used as a means to integrate and expose both capabilities offered by end-devices and EdgeX middleware services to application designers. Additionally *Process DSLs* are used as a means to foster reuse of medium and large-grained business logic as reusable features across applications.

The native and process DSLs have been previously applied also to robotic scenarios [19], and [18] shows how to craft REST services and cloud services in

the DIME environment. In [15] we also adopted the ADD-Lib [12] for policy definition. The policies designed with ADD-lib translate to efficient code that is integrated in DIME through its Native DSL mechanism.

Altogether, this shows flexibility, ease of extension, support of high-assurance software quality, agility, security, a service-oriented approach, and containerization, as well as proven compatibility with EdgeX. The contributions of this paper concern the portability of the architecture, methodology, and the reuse of many artefacts also in the Predictive Maintenance domain.

3 Industrial Automation Use Case: Predictive Maintenance

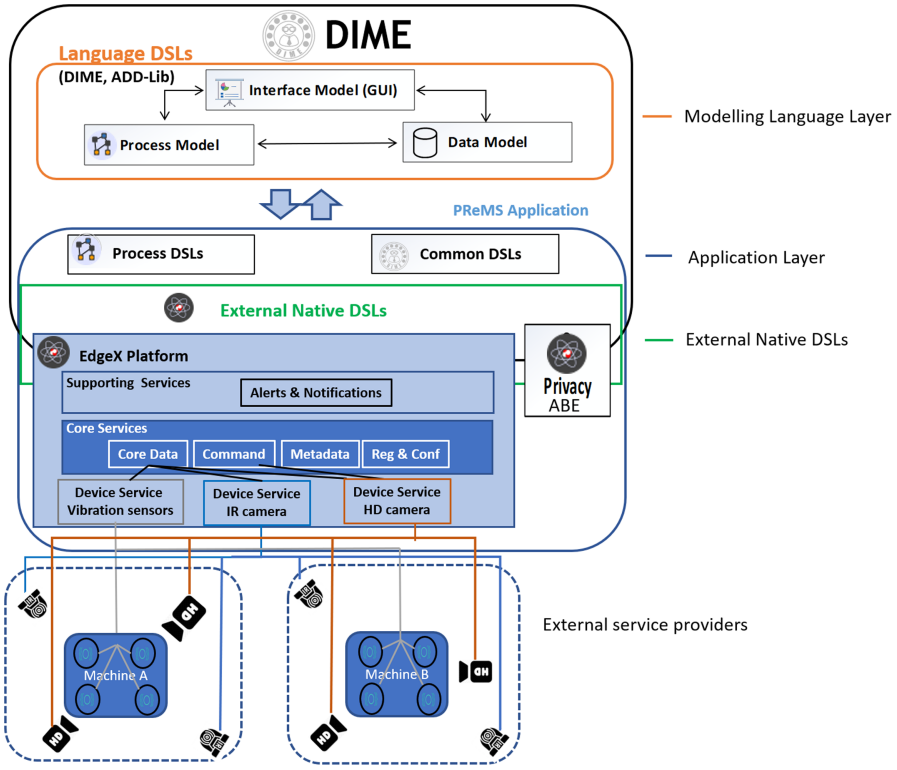

Fig. 2. EdgeX along with DSLs for an IIoT use case - predictive maintenance system (PreMS)

We illustrate the approach with a simple industrial automation use case concerning predictive maintenance, shown in Fig. 2. We consider a monitoring system to monitor machine health, whose purpose is to make decisions on machine maintenance, avoiding unscheduled downtime and preventing machine failure.

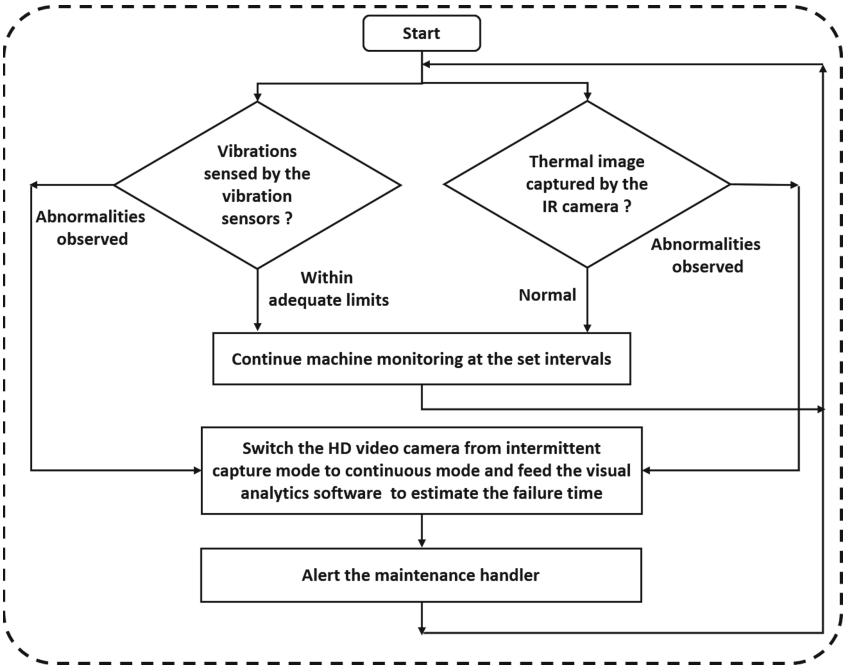

Fig. 3. Predictive maintenance system - an excerpt from the application logic

The monitoring system consists of *wireless vibration sensors* attached to the machine in key locations, an *infrared camera* and a *high definition video camera* that monitor the machine from a short distance and deliver image data for visual analytics to identify machine states. The sensors are wirelessly connected using Bluetooth technology, and the cameras are connected using Wi-Fi (infrared camera) and 5G (high definition camera) technologies. The *edge analytics* collecting and analysing data creates maintenance alerts to machine handlers indicating machine state and likely failure time. This allows the machine handlers to schedule planned downtime of particular machines avoiding machine failure and disrupting the manufacturing process. The general form of the platform architecture shown in Fig. 2 is detailed in [18]. Here we highlight the two layers of DSL and the specific PReMS instance of the External layers: Native DSLs and service providers.

Figure 3 details a high-level overview of the application logic of the Predictive Maintenance System (PreMS). The vibration sensors and the thermal imaging (infrared) cameras associated with each machine periodically report their data to the decision system. Under normal conditions, the HD cameras operate in an intermittent mode so that the industrial wireless network is not flooded with high volumes of unwanted data. Suppose any abnormalities are observed either in the machine vibrations or in the thermal conditions detected by the IR camera. In that case, PreMS switches the HD video camera from intermittent mode to

continuous operating mode for further analysis and uninterrupted video frames are fed to the visual analytics software for a detailed investigation of the scenario. Accordingly, various alerts are sent to different maintenance handlers based on the current machine state and its estimated failure time.

The sensors and cameras are connected to an EdgeX edge deployment that deals with the heterogeneity in connectivity and sensing modalities as well as to provide management capabilities to the monitoring system. We now look at the stages involved in provisioning the preventative maintenance system (PreMS) using the core and device layers of EdgeX, and DIME for EdgeX integration and to design the PreMS application. We detail below the three main stages involved from the point of view of a PreMS developer. We assume that EdgeX runs on an on-premise edge computer.

3.1 Provisioning Devices in EdgeX: The Integration Layer

The device service layer of EdgeX (Fig. 1) aids the sensor/camera provisioning process through protocol-specific device connectors (e.g. BLE, MQTT, etc.) in the device service layer. It also supports the development of custom device services using an available SDK. The device service layer converts the data generated from various types of devices with different protocols into a common EdgeX data structure and sends the data to other microservices through REST calls. As shown in Fig. 2, our PreMS use case consists of three device services; one for each of the device protocol types (Vibration sensor-BLE interface, IR camera - WiFi interface, HD camera - 5G interface). A device service is only aware of the generic communication protocol, and the specific details about a particular device are uploaded to EdgeX through device profiles. The device specific details include the sensor data types, the sensing and actuation commands supported by the device (REST API calls) in addition to the generic information such as the manufacturer, model number, etc. The detailed process of registering an external device to EdgeX through its device profile for a building automation use case is detailed in [15]. In our case the procedure is the same: the *PreMS Native DSL* is an external Native DSL (see Fig. 2) for the devices encompasses a DSL per each IoT *device type*, and also here, the ability to use device profiles enables a very easy commissioning of multiple *device instances* of the same kind, e.g. the HD cameras for Machine A and Machine B in Fig. 2 as two instances of the HD camera device type.

As shown in Fig. 2, the Native DSLs expose to DIME collections of basic services across one or more application domains. They comprise “atomic” services which are implemented in code or as calls to external services, APIs and platforms. Once the devices are provisioned and connected to EdgeX, monitoring data can flow to EdgeX. Each device sends its data to the core-data service through its associated device services in the form of events/readings. The sensed data are stored in the database and made available on the common message bus (optional). The other microservices (e.g., Rules Engine) can operate on the data and derive local decisions based on various policies.

3.2 Data and Processes: The Application Layer

The PreMS application data model in DIME refers to the EdgeX entities, to the various device profiles, and includes the other entities relevant to the PreMS application, like users, locations of the machines and policies for later decision making. For every elementary or complex object in the data model, a set of services are automatically created in DIME (get and set services), so that the definition of an object and its attributes, as well as of an enumeration type, automatically produces a DSL for its management coherent with its structure and types without need of manual coding. This is particularly useful when we consider industrial users who are mostly not software developers.

At the application level, as shown in Fig. 2, DIME comprises of its own basic DSL, the *Common SIBs library*, with generic capabilities to handle files, decisions, comparisons, iterations, and more. It also provides a rich DSL for GUI element design. DIME addresses primarily Web applications, so GUI models using the design and functional elements provided in the GUI DSL are the way to define the look and feel of web pages, as well as their navigation structure.

Typically, applications have a hierarchical architecture, because processes can include other processes, therefore there are entire *Process DSLs* arising bottom-up from the design and sharing of processes for reuse in other processes. The process symbol is that of a graph because the processes (or workflows) are modelled as hierarchical graphs that define both control flow and data flow of the application. The Native DSLs on the contrary comprise of "atomic" services, hence the atom symbol on the icons. For example, the *UploadDevProfile, CreateDev* and *StartDev* services shown in Fig. 4 are atomic services belonging to the EdgeX native DSL in DIME, and the entire process in Fig. 4 implements the *SetupDevice* process, which is a process.

Figures 4 and 5 show how control flow and data flow are represented: the workflow logic is the control flow, denoted by solid arrows (e.g. from *Start* to *UploadDevProfile* in Fig. 4), and the data flow is denoted by dotted arrows, that connect directly the data elements (e.g., the output *devTypeID* produced by *UploadDevProfile* that is passed as input to *CreateDev* in Fig. 4), or refer in input or output to the Data context, as shown in Fig. 5 on the left, especially when complex data types are involved. Complex data types are typically created at once, but read or modified field by field in the process workflow. This is shown for the *device_type* record: its field *DeviceTypeID* is used (i.e., read) by three services but its *List_of_Services* field is used only by the *SendCommand* service.

3.3 Reuse Through DSLs

In the platform approach inherent to DIME and EdgeX, *reuse* of small and large components across applications in a domain and also across domains is an essential goal of the platforms, and a key benefit for the users. Referring again to the architecture in Fig. 2, the internal functionality offered by EdgeX is represented in DIME as a native palette to the application designers. The same applies to the EdgeX Core layer services, to the CP_ABE security service Native DSL for our

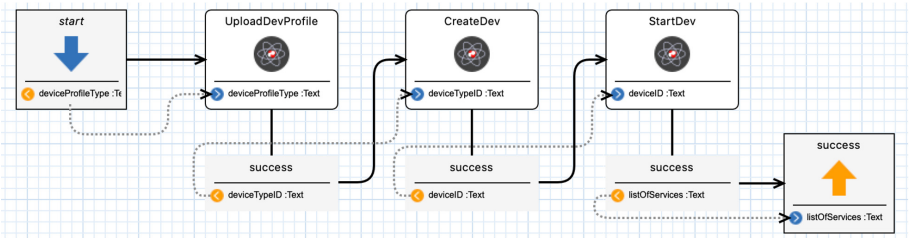

Fig. 4. EdgeXBAu processes: the SetupDevice process in DIME registers the device type and physical device to EdgeX

Attribute Based Encryption capabilities, as well as for other collections of services in the Common SIBs library, that include generic capabilities e.g. to handle files, decisions, comparisons, iterations, and more. In this sense, the "layers" in DIME differ from the layers in EdgeX.

From an EdgeX point of view, the granularity of services it provides in the four layers in Fig. 1 matters, and there is a clear hierarchy among the different service libraries at different levels. This is a service provider point of view, which refers to granularity and position in the internal service provision architecture.

For DIME, however, the point of view is that of the user, i.e. the consumer of services – whatever is external to the application layer is an external service, i.e. a "black box" seen as a *unit of functionality* ready to be used as a ready-made atomic entity. The important concept here is the unit – seen from DIME, any service provening from EdgeX is *atomic*, independently of their EdgeX-internal structure and complexity. This is a very powerful abstraction – not only EdgeX, but any external source of functionalities is treated the same way as is for example the UR3 robot command language in [19] or the Kinect and ROS related libraries in [11]. This virtualization and hiding mechanism is essential for a platform, in order to master heterogeneity and interoperability, so that DIME allows to mix and match services provening from different application domains, and service providers.

The important concepts are the *native DSLs* for the basic functionalities and the *process-level DSLs* for the hierarchical construction of applications within DIME. All the Common SIBs and the GUI DSL come with DIME, so they are written once but available to any application. DIME's EdgeX DSLs are also shareable across all the DIME applications that use EdgeX. The degree of generality and genericity in the specific domain is key to the degree of reusability. For example, our PreMS application "inherits" from the BAu application of [15] the collection of DIME Processes produced during the BAu application design that are specific to EdgeX, but not to BAu. Concretely, Figs. 4 and 5 show two such processes:

– Figure 4 shows the *SetupDevice process* in DIME, which registers the device type and physical device to EdgeX, originally called process *BAuSetupDevice* in [15], and

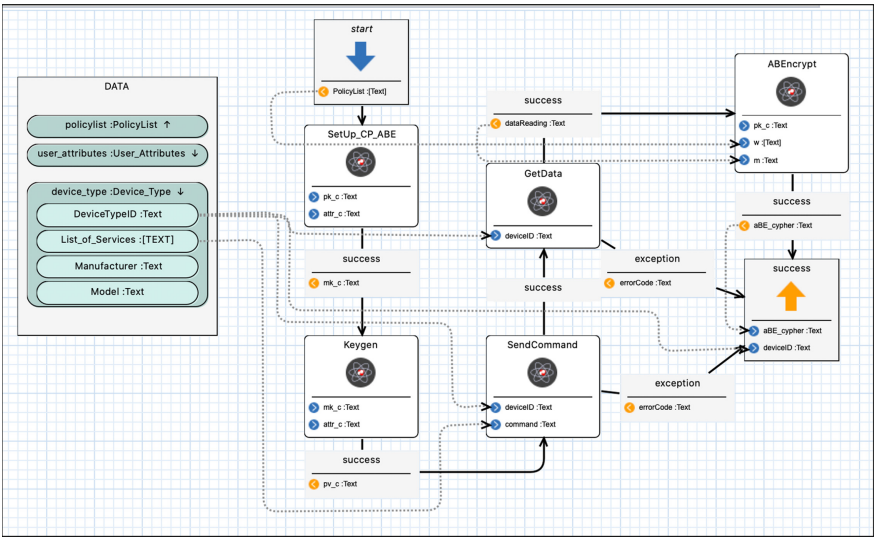

Fig. 5. EdgeX processes: simplified EdgeXOperations service

– Figure 5 shows the simplified EdgeXOperations service, which operates any specific application's system (originally the process BAuOperations in [15], or at any time allows users to decide to reconfigure, restart or stop the application and terminate.

Many other processes are application specific, thus do not carry over from a preexisting case study to the PreMS, and we have to design our own. Similarly, many native DSLs of the BAu application concern specific IoT devices (like PIR sensors, CO2 sensors etc.) not relevant to the PreMS application, so we have to design our own native DSL for the specific device types we use – vibration sensor, IR camera and HD video camera.

An MDD approach as supported by DIME through the models followed by code generation of the resulting application makes this reuse much easier and more intuitive than if we had to understand and reuse manually produced code.

3.4 Handling Alerts and Machine Failures

Here we detail the various stages involved in handling the alerts or machine failure scenarios. Once the PreMS detects any abnormal working conditions for any machine, alerts/notifications are sent to maintenance handlers for proper maintenance. Figure 6 shows the high-level internal architecture of the alerts and notification microservice provided by EdgeX as part of its *Alerts & Notification* library of the *Supporting Services* level (see Figs. 1 and 2). The *Notification Handler* receives the request to send out alerts or notifications from other microservices or applications (on-box/ off-box) through APIs of different application protocols (REST, AMQP, MQTT - shown on the left side of Fig. 6). In the

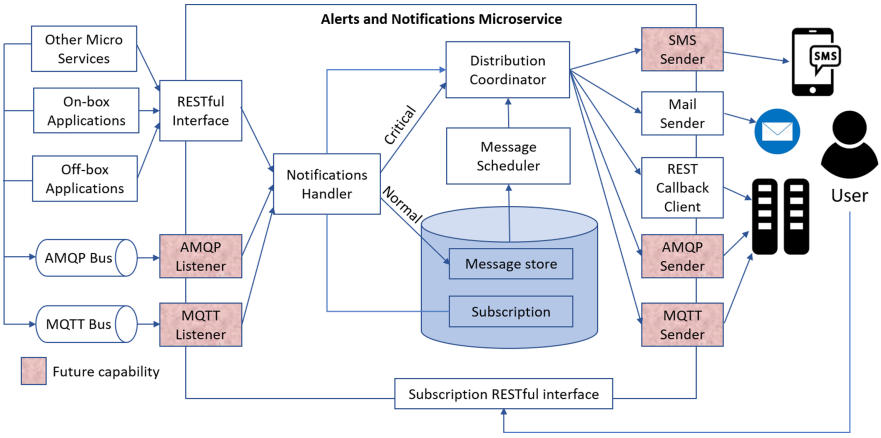

Fig. 6. High-level architecture of alerts & notifications microservice within EdgeX [9]

considered use case scenario, the alerts/notifications may be initiated either by the device service/ rules engine (when the vibration sensor readings fall outside the expected values), or by the thermal image inspection software (when any of the machine or its parts gets overheated), or by the visual analytics software. The *Notifications Handler* persists the received notification and hands it over to the *Distribution Coordinator*. The *Distribution Coordinator* queries the Subscription database to get the intended recipient details of the particular notification, including the communication channel information such as SMS, email, or API destination endpoint (REST, AMQP, MQTT). Accordingly, the Distribution Coordinator passes the alert/notification to the corresponding channel sender (shown on the right side of Fig. 6), which sends out the alert/notification to the subscribed recipients.

For us, at the application level in DIME, this is just yet another atomic service provening from EdgeX.

4 Secure Access Policy for PreMS

In the PreMS application, we use the EdgeX secret store security feature for storing sensitive data, while secure access of data is performed using the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [4]. For user authentication, we use the EdgeX API gateway security element. For the industrial automation application, we define a secure access policy mechanism as follows: we consider four attributes involved in our use case scenario, namely Maintenance Handler (MH), Mechanical (ME) department, Video Analysis (VA) department and Decision Management System (DMS), whose main functionalities are described as follows: (i) the MH of ME department is responsible for monitoring the health of the machine and have access to the data collected by the vibration sensor; (ii) the MH of VA department analyses the images captured by the infrared camera

and the high definition video camera, and notifies the DMS Maintenance Handler if any deviation from normal behaviour is noticed; and (iii) the MH of the DMS takes the final call for the need of generating an alarm if any emergency situation occurs and notifying the same to the PreMS for further actions.

Table 1 shows the different types of devices used, the type of data they generate and the access policies. The access policies define which entities have access to specific data generated by the devices. As mentioned previously, we intend to allow fine-grained secure access control based on attributes, and for this we leverage public-key encryption, i.e., CP-ABE. In CP-ABE, the ciphertexts are identified with access policies and the private keys with attributes. Therefore, a message encrypted using CP-ABE produces a ciphertext which can be decrypted using a private key by a user who owns a set of attributes and satisfies the access policy. One of the key features of CP-ABE is that it enables the definition of top-level policies and is particularly suitable in settings, where it is necessary to limit the access of a particular information only to a subset of users within the same broadcast domain [2].

The CP-ABE scheme consists of the following four algorithms:

- SETUP(). The algorithm takes no input other than the implicit security parameters and returns the public key pk_c and master key mk_c.
- KEYGEN(mk_c, $Attr_c$). The key generation algorithm takes mk_c and the user attribute list $Attr_c$ as inputs and returns a private key pv_c.
- $\text{ABE}_{pk_c,w}(m)$. The encryption algorithm takes pk_c, an access policy w over the pool of attributes, and sensor reading m as inputs. It returns a ciphertext that can only be decrypted by a user that possesses a set of attributes $Attr_c$ such that $Attr_c$ satisfies w.
- $\text{ABD}_{pv_c}(c)$. The decryption algorithm takes pk_c, pv_c and the ciphertext c as inputs. It outputs the plaintext m if and only if the user $Attr_c$ satisfies w.

Table 1. Device type and access policy

Device type	Data type	Access policy
Wireless Vibration Sensor	[Integer/Float]	MH $\wedge$ (ME $\vee$ DMS)
Infrared Camera	[H.264/ numeric array]	MH $\wedge$ (VA $\vee$ DMS)
High Definition Video Camera	Float [H.264/ numeric array]	MH $\wedge$ (VA $\vee$ DMS)

In order to implement this, we profit again from the work done in [15]: as the four algorithms are domain and application independent, we reuse the Native DSL for CP-ABE they produced, as well as the process in Fig. 5. What changes are the specific access policies for the PreMS from Table 1, which we will again define using the low code model driven development tool ADD-Lib [12] and the surrounding PreMS application logic.

5 Conclusions

We have shown how EdgeX simplifies integrating and managing a wealth of IoT devices and protocols that are central to applications in cyber physical manufacturing systems like predictive maintenance. The platform character of EdgeX and of modern low-code application design environments (LCADEs) is central to their ability to enable high reuse of existing resources, like microservices, components, and algorithms (e.g. CP-ABE), embedded through a Native DSL mechanism in DIME, our chosen LCADE. The DSL concept is central to the integration, the virtualization and the reuse. For example, the extension of existing middleware service platforms like EdgeX to include advanced security mechanisms like CP-ABE is made easy as CP-ABE is in DIME a native DSL plus a collection of domain and application independent processes. The DSLs also support application extension and evolution with minimum programming effort.

Because DIME adopts a generative approach to code, every time the models are modified or extended, the code is efficiently re-generated and redeployed, in a DevOps fashion. The consequence is that every version of the deployed code is "clean": it contains only what is needed (minimal), it contains no patches (it is most recent), and it is unspoiled by human intervention, which is known to inadvertently introduce bugs when fixing detected issues.

In terms of lessons learned, an integrated modeling environment like DIME is indeed superior to its predecessors, that addressed only the Native DSLs for integration and the processes. For example, in [16] the jABC tool, also a low-code and generative platform, while it supported a SIB palette for the commands available to steer a Lego robot, serving the same purpose as the Native DSLs, the data model was not supported in an integrated way, nor there was a possibility to define GUIs. We see the ability to work on all these design facets within the same environment as a clear asset. It eases adoption and better supports a multidisciplinary collaboration with experts of other domains.

From the point of view of modelling styles, in the CPS and engineering domain the prevalent approach is based on simulations or on hybrid models, e.g. for various kinds of digital twins that serve as virtual replicas of CPS. In the track dedicated to the engineering of Digital Twins for Cyber-Physical Systems in ISoLA 2020 [?], for example, providing a recent panorama of modelling approaches and applications in areas close to our own research, the considered models are predominantly quantitative, answering questions about uncertainty, precision and tolerance. Even when the applied model-based and formal techniques support some form of reasoning, this happens mostly in a statistical and AI or AI-like fashion. We concentrate here instead on a fully MDD approach to application design, which is still new to the CPS domain. Its relation with the Digital Twin concept is addressed in detail in [19], and the role of formal models for the low code, MDD application design as used here vs. a corresponding digital twin generated via active automata learning is discussed in [20].

The specific support of evolution is very attractive for our specific setting: our long term objective is to produce a collaborative design ecosystem and an open virtual testbed for intra-but also interorganizational advanced manufacturing, where we expect solutions to grow and evolve over time. We could envisage

the EdgeX Distribution Coordinator shown in Fig. 6 to interface in the future with an advanced specialized alarm and notification escalation solution, like the Enterprise Alert product (Derdack, n.d.), and this extension should happen with minimum coding, minimum disruption to the underlying PreMS application, minimum effort, including testing, and minimum risk. We therefore value the simplicity, reuse, openness, and abstraction that these platforms jointly provide.

Acknowledgment. This project received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Smart 4.0 Co-Fund, grant agreement No. 847577; and a research grant from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (CONFIRM Centre)

References

1. Aceto, G., Persico, V., Pescapé, A.: A survey on information and communication technologies for industry 4.0: state-of-the-art, taxonomies, perspectives, and challenges. IEEE Commun. Surv. Tutor. **21**(4), 3467–3501 (2019). https://doi.org/10.1109/COMST.2019.2938259
2. Ambrosin, M., Busold, C., Conti, M., Sadeghi, A.-R., Schunter, M.: Updaticator: updating billions of devices by an efficient, scalable and secure software update distribution over untrusted cache-enabled networks. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 76–93. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_5
3. Antonini, M., Vecchio, M., Antonelli, F.: Fog computing architectures: a reference for practitioners. IEEE Internet Things Mag. **2**(3), 19–25 (2019). https://doi.org/10.1109/IOTM.0001.1900029
4. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of IEEE S&P, pp. 321–334 (2007)
5. Bonavolontà, F., Tedesco, A., Moriello, R.S.L., Tufano, A.: Enabling wireless technologies for industry 4.0: state of the art. In: 2017 IEEE International Workshop on Measurement and Networking (M N), pp. 1–5 (2017). https://doi.org/10.1109/IWMN.2017.8078381
6. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
7. Christou, I.T., Kefalakis, N., Zalonis, A., Soldatos, J., Bröchler, R.: End-to-end industrial IoT platform for actionable predictive maintenance. IFAC-PapersOnLine **53**(3), 173–178 (2020)
8. Confirm: Confirm smart manufacturing - Science Foundation Ireland Research centre. https://confirm.ie/
9. EdgeX Foundry: "EdgeX Alerts and Notifications". https://docs.edgexfoundry.org/2.0/microservices/support/notifications/Ch-AlertsNotifications/
10. EdgeX Foundry: "Why EdgeX". https://www.edgexfoundry.org/why_edgex/why-edgex/
11. Farulla, G.A., Indaco, M., Legay, A., Margaria, T.: Model driven design of secure properties for vision-based applications: A case study. In: T.Margaria, M.G.Solo, A. (eds.) The 2016 International Conference on Security and Management (SAM 2016). Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application", pp. 159–167. CREA Press (2016)

12. Gossen, F., Margaria, T., Murtovi, A., Naujokat, S., Steffen, B.: DSLs for decision services: a tutorial introduction to language-driven engineering. In: ISoLA 2018, Proceedings, Part I, pp. 546–564 (2018). https://doi.org/10.1007/978-3-030-03418-4_33
13. Han, K., Duan, Y., Jin, R., Ma, Z., Rong, H., Cai, X.: Open framework of gateway monitoring system for internet of things in edge computing. In: 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), pp. 1–5. IEEE (2020)
14. IoTech: "IoTech The Edge Software Company". https://www.iotechsys.com/
15. John, J., Ghosal, A., Margaria, T., Pesch, D.: DSLs for model driven development of secure interoperable automation systems. In: Forum on Specification & Design Languages (Accepted for Publication) (2021), (in print)
16. Jörges, S., Kubczak, C., Pageau, F., Margaria, T.: Model driven design of reliable robot control programs using the jABC. In: Proceedings of 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2007), pp. 137–148 (2007)
17. Kim, J., Kim, C., Son, B., Ryu, J., Kim, S.: A study on Time-series DBMS application for EdgeX-based lightweight edge gateway. In: 2020 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1795–1798. IEEE (2020)
18. Margaria, T., Chaudhary, H.A.A., Guevara, I., Ryan, S., Schieweck, A.: The interoperability challenge: building a model-driven digital thread platform for CPS. In: Proceedings ISoLA 2021, International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Rhodes, October 2021. Lecture Notes in Computer Science, vol. 13036. Springer (2021)
19. Margaria, T., Schieweck, A.: The digital thread in Industry 4.0. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) IFM 2019. LNCS, vol. 11918, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34968-4_1
20. Margaria, T., Schieweck, A.: The digital thread in Industry 4.0. In: Olderog, Ernst-Rüudiger, S.B., Yi, W. (eds.) Model Checking, Synthesis and Learning. Lecture Notes in Computer Science, vol. 13030. Springer (2021)
21. Milić, S.D., Babić, B.M.: Toward the future-upgrading existing remote monitoring concepts to IIoT concepts. IEEE Internet Things J. **7**(12), 11693–11700 (2020). https://doi.org/10.1109/JIOT.2020.2999196
22. Paniagua, C., Delsing, J.: Industrial frameworks for Internet of Things: a survey. IEEE Syst. J. **15**(1), 1149–1159 (2021)
23. Wollschlaeger, M., Sauter, T., Jasperneite, J.: The future of industrial communication: automation networks in the era of the Internet of Things and Industry 4.0. IEEE Ind. Electron. Mag. **11**(1), 17–27 (2017). https://doi.org/10.1109/MIE.2017.2649104
24. Xu, R., Jin, W., Kim, D.H.: Knowledge-based edge computing framework based on CoAP and HTTP for enabling heterogeneous connectivity. Pers. Ubiq. Comput. 1–16 (2020)
25. Xu, R., Jin, W., Kim, D.: Microservice security agent based on API gateway in edge computing. Sensors **19**(22), 4905 (2019)
26. Zhang, J., et al.: A blockchain-based trusted edge platform in edge computing environment. Sensors **21**(6), 2126 (2021)