Context-Aware Digital Twins to Support Software Management at the Edge

Rustem Dautov[1 [0000−0002−0260−6343]] and Hui Song[1 [0000−0002−9748−8086]]

SINTEF Digital
Oslo, Norway
{rustem.dautov,hui.song}@sintef.no

Abstract. With millions of connected edge gateways, there is a pressing challenge of remote maintenance of containerised software components after the initial release. To support remote update operations, edge software providers have been increasingly adopting digital twin-based device management platforms for run-time monitoring and interaction. A common limitation of these solutions is the lack of support for modelling the multi-dimensional context of edge devices deployed in the field, which hinders the software management in a tailored and context-aware manner. This paper aims to address this lack of context-awareness in digital twins required for edge software assignment by introducing two modelling principles, which allow focusing on the device fleet as a whole and capturing the diverse cyber-physical-social context of individual devices. As part of proof of concept, these principles were incorporated in an existing digital twin platform. This prototype implementation demonstrates the viability of the proposed modelling principles via a running example in the context of a telemedicine application system.

Keywords: Digital Twin · Context Awareness · Edge Computing · IoT · Device Fleet · Eclipse Ditto · Remote Patient Monitoring.

1 Introduction

With the increasing computing and networking capabilities, IoT devices and edge gateways have become part of a larger IoT-edge-cloud computing continuum, where processing and storage tasks are distributed across the whole network hierarchy, not concentrated only in the cloud. At the same time, this also introduced continuous delivery practices to the development of software components for network-connected edge gateways. These devices are placed on end users' premises and are characterised by changing multi-dimensional contexts, forcing developers to maintain multiple software versions and frequently re-deploy them on a distributed fleet of devices with respect to their updated contexts. Unlike the traditional cloud model, where computing resources are homogeneous and concentrated in a single physical location, an edge fleet may be distributed across thousands of heterogeneous devices, each with a unique context in terms of hardware capacity, surrounding physical environment, network connection, user preferences, *etc.* To address such heterogeneous contexts, software components are also becoming increasingly diverse, and edge software providers often

simultaneously maintain multiple active versions of the same application. Taken together, the increasing diversity on both sides, *i.e.* edge devices and software components, raises a challenge of *assignment* implemented in a precise, reliable and scalable manner.

Doing this correctly and efficiently goes beyond the manual capabilities of software vendors and requires an intelligent and reliable automated solution. This has given rise to device management cloud platforms, which allow collecting information from distributed devices and provide a near real-time view on the overall fleet. This functionality is often underpinned by the prominent concept of *Digital Twins* (DTs). However, existing platforms often require increased coding and modelling effort from edge software providers in order to implement software assignment for the two main reasons. First, the existing solutions typically focus on individual devices in isolation from each other, thus neglecting their interrelations. Second, the default DT modelling support mainly considers the traditional context metrics, such as hardware and software resources, to assign software updates, neglecting a much more diverse and dynamic multi-dimensional context of each edge device. As a result, this lack of modelled context information forces edge application providers to manually map multiple versions of their software, which eventually becomes a bottleneck in their agile development processes.

Accordingly, this paper aims to address the following high-level research question – **how to model the dynamic multi-dimensional device context in order to support correct and targeted assignment of software management updates on edge infrastructures?** The contribution of the paper is threefold. First, by exploiting the DT concept we elevate the level of modelling abstraction to the *fleet level*, as opposed to the state of practice primarily focusing on individual devices. Second, we enhance digital representations of devices within a fleet with a notion of *multi-dimensional context*, as opposed to the existing IoT device management platforms primarily focusing only on hardware and, occasionally, software aspects. Third, we demonstrate the applicability of the proposed methodology with a proof of concept from the telemedicine domain.

The rest of the paper is organised as follows. Section 2 provides motivation behind this research by explaining the challenges existing in the edge software management. Section 3 briefs the reader on the state of practice and related research works in the domains of DTs and software assignment on edge infrastructures, highlighting the existing limitations. Section 4 explains how the prominent DT paradigm can be applied to address these limitations. Section 5 puts theory into practice and describes how the proposed research ideas have been implemented on top of the existing DT platform Eclipse Ditto. Section 6 summarises the results and concludes the paper with an outlook for future work.

2 Research Context and Motivation

2.1 Software Maintenance at the Edge

The ubiquitous connectivity has given rise to a new sector within the ICT market comprised by companies specialising in software development for cloud-

connected edge infrastructures. Typically, such software providers are not directly involved in hardware manufacturing, but rather rely on some third-party gateways shipped directly to end users, along with various IoT sensors and actuators. While downstream IoT devices are not always Internet-connected and are not equipped with sufficient computing resources, edge gateways have traditionally served to collect sensor data, transfer it to the central cloud back-end, and relay actuation commands back to actuators, as illustrated by Fig. 1.

More recently, edge gateways have also become fully-functional processing units in their own right, going beyond the passive transferring to the cloud. Indeed, edge gateways are equipped with relatively powerful computing (*i.e.* CPU and RAM) and networking capabilities to run some business applications on top of data collected from downstream IoT sensors. Thanks to these increased capabilities, edge devices are not only able to support intensive data exchange between the IoT and the Cloud layers, but also run complex software on top of Linux OS, including a user interface for direct interaction with end users. With the recent advancements in virtualisation and containerisation technologies, edge software is often packaged and released as Docker containers – the *de facto* standard for enterprise-level software containerisation. The hardware and software counterparts of the described edge-based installations comprise the next generation of connected cyber-physical systems across a wide range of intelligent scenarios, such as various smart spaces, transportation, Industry 4.0, telemedicine [10], *etc.*[1]

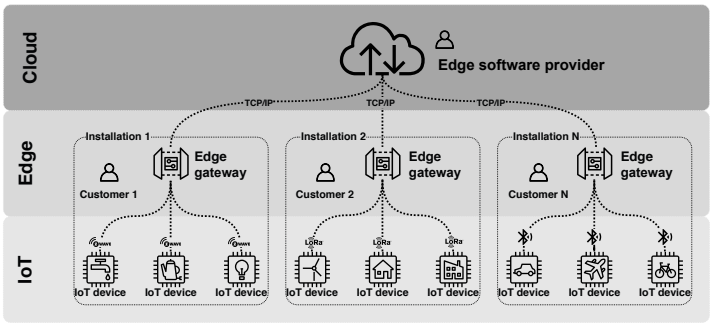

Fig. 1. Application system spanning across the Cloud-Edge-IoT infrastructures.

Like all software, edge applications may contain bugs or security flaws, which will only emerge after the initial deployment. At the same time, software providers also frequently release functional updates in incremental manner following agile development practices. Unlike in traditional data centres, maintenance of edge software needs to take place remotely, since edge devices are installed and connected on end users' premises. For a relatively small fleet of devices, this might

[1] Throughout this paper, and in Section 5 in particular, we will use telemedicine as the main reference example. Nevertheless, the described concepts apply to other scenarios dealing with edge software management.

not be a challenge. Larger fleets, comprised of thousands of units distributed around the globe, need an automated device management platform to remotely provision, monitor, reset or lock every device, as well as to push updates whenever a new software version is released. Automating functions through a device management platform saves time and resources, and minimises human errors. Examples of such platforms include mainstream commercial solutions such as Azure IoT Hub[2] and AWS IoT Greengrass,[3] niche market offerings such as Balena Cloud,[4] and open-source community-driven toolkits such as Eclipse hawkBit.[5] All these platfroms focus on container-based software management and rely on some device-side monitoring agents for collecting metrics at run-time.

2.2 Challenge: Context-Awareness for Software Assignment

Assignment is an important part of all software management activities, be it the initial release or the follow-up maintenance. Unlike the centralised cloud model, where computing resources are relatively homogeneous, a fleet of edge gateways consists of distributed and heterogeneous devices, which have diverse multi-dimensional contexts in terms of hardware capacity, network connection, physical deployment, user preferences, *etc.* [8]. Developers often need to maintain several software variants to fit such different device contexts. For example, some devices may require a variant tailored to a specific hardware capacity, while some others with limited connectivity may need a variant with lower bandwidth requirements. This can be formulated as as a generic assignment problem – *i.e.* how to assign m software variants to n devices, so that each device is assigned with a variant that matches its context. At the same time, the whole fleet may also need to meet its global goals, *e.g.* maximise software diversity in the fleet for security purposes [16] or pick a sub-set of devices for preview and testing.

The existing tag-based fleet assignment mechanisms offered by device management platforms are sufficient for rather simple scenarios with a few device properties to take into account. Devices can be logically grouped so that a particular maintenance update is applied only to a specific group. For example, grouping can be based on physical location or hardware architecture. However, this is not expressive enough to address software assignment scenarios with hundreds and thousands devices, each having its own continuously changing multi-dimensional context – on the one hand, and multiple software versions – on the other. In these circumstances, correctly assigning software components in a precise and scalable manner heavily depends on up-to-date contextual information about managed devices, which seems to be not immediately available in existing solutions, which rely on general-purpose monitoring agents collecting a limited number of hard-coded performance metrics. In the absence of richer context-aware information about the managed fleet, using the default available

[2] `https://azure.microsoft.com/products/iot-hub/`
[3] `https://aws.amazon.com/greengrass/`
[4] `https://www.balena.io/`
[5] `https://www.eclipse.org/hawkbit/`

tag-based assignment mechanisms is still possible, but would result in a complex collection of chained fine-grained $if-then$ conditions – an error-prone and difficult-to-maintain solution.

3 State of the Art and Practice

3.1 Digital Twins for Edge Fleet Management and Context Modelling

As already explained, by using a device management platform, edge application providers are able to remotely provision, monitor, and maintain their device, as well as push software updates whenever a new version is released. In recent years, all these tasks have been underpinned by the prominent concept of DTs. DTs are virtual models designed to accurately reflect physical objects equipped with various sensors related to certain aspects of their functionality. They enable to create rich digital models of anything physical or logical, from simple assets or products to complex cyber-physical environments. The collected sensor data is relayed to a processing system and applied to the DT. Once updated with such data, the DT can be used to run simulations, explore performance issues, and identify possible improvements – which can then be applied back to the original physical twin using bi-directional IoT connections.

Among many research efforts on DTs applied to the IoT [20,21] an important part belongs to research on DT interoperability and standards [14,22]. An industry-driven effort in this context was done by Microsoft's DT Definition Language (DTDL)[6] to provide more semantics to modelled physical environments. Albeit open-source, it can primarily be used only within the Azure ecosystem. A similar attempt was made by the now-retired Eclipse Vorto.[7] W3C Web of Things (WoT) has come up with another effort to standardise the modelling domain of IoT systems its Things Description model [6] – a formal information model and a common representation to describe the metadata and interfaces of IoT devices. In parallel, there is also an on-going effort to integrate the Semantic Web ontologies to provide more formal semantics and reasoning to the WoT.

Another relevant research field is *context modelling*, which has already been active for a few decades [5,15]. Context modelling produces a formal or semi-formal description of the context information to create a context-aware system. Some recent attempts [11,7] have started considering the cyber-physical dimension of the IoT and edge environments. In particular, there is a demand for context-awareness to assess trustworthiness of devices [18], as well as to complement application-specific analytics with enriched context information, *e.g.* disease diagnostics based on extra information from IoT sensors [12].

In summary, engineering of context-aware DTs is currently *ad-hoc* to a great extent, which is a challenge for quality-controlled development, deployment, and operation. Most works focus on a limited set of conventional metrics, not sufficient and suitable for edge software maintenance scenarios. A common limitation

[6] `https://learn.microsoft.com/azure/digital-twins/concepts-models`
[7] `https://www.eclipse.org/vorto/`

of the existing approaches is the hard-coded focus only on modelling the cyber-aspects of the managed physical entities, while more extensive modelling and coding is left to system administrators to be done manually.

3.2 Software Assignment

The existing works on software assignment at the edge primarily focus on resource usage optimisation [19,13,4,17], in which the objective function is typically based on generic infrastructure-level metrics, such as network latency [25], hardware resources utilisation, energy consumption, *etc.* and less frequent metrics, such as CO_2 footprint, monetary costs, number of tenants, *etc.* Optimisation metrics and related challenges are extensively surveyed in [2]. In all these works, edge software is typically treated as a black box, owned or managed by some external third party. As opposed to this, this proposed work tackles the software assignment problem from the perspective of an application provider, who owns and manages the whole vertical application system, including edge and cloud components. This means that software placement is mainly driven by application- and business-level requirements and evaluated against continuously monitored device-specific contexts, not strictly limited to the infrastructure level.

In this light, implementing software assignment requires modelling the application domain (*i.e.* software requirements and device context) and corresponding multi-dimensional constraints. Research approaches to solving complex constraints have resulted in several theories and tools, such as Satisfiability Modulo Theories (SMT) [1], which offered several fully automated fleet assignment solutions based on various constraints and conditions. The SMT-based approaches are specifically popular and efficient due to their expressive and rich modelling language [3,23,24]. Nevertheless, they still require manually modelling the target system with some contextual information and testing the constraints.

Managing software updates across a large fleet of devices is an everyday task for the leading mobile OS providers, such as Apple and Google. At present, their adopted over-the-air update mechanism implements a publish-subscribe model, where mobile devices first get notified of and then fetch available updates through the centralised marketplace. Since smartphone fleets are rather homogeneous in terms of hardware and OSs, there is no challenge of multi-criteria software assignment as such, and the compatibility check is performed only once, upon the initial installation. Furthermore, in a fleet of smartphones there are typically no global system goals, such as even distribution of components or A/B testing.

4 Modelling Principles for Context-Awareness

As we have argued, existing approaches lack the support for context-aware software assignment at the edge, which we aim to tackle by this paper. We now proceed with a description of the two main underpinning design principles.

I. Device fleet as the main level of abstraction. This design principle represents one step up from the conventional approach to maintain DTs of individual devices. By lifting the abstraction level, we are thus able to maintain a

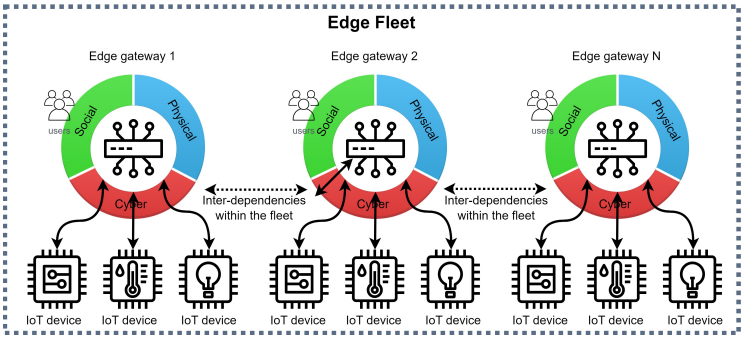

Fig. 2. Multi-dimensional context of edge gateways within a fleet.

live view not only on the managed individual devices, but also on the dynamic inter-relations between them – something that can potentially play an important role when deciding what software management commands need to be applied to specific devices. This is especially important when considering some global software management policies within a fleet, where assigning a software version to one device will depend on what version has already been assigned to another. For example, in DevOps, A/B testing is often implemented to test newly-added features on a limited sub-set of staging devices before pushing them to production. Another example is the artificial *software diversity* – *i.e.* a common technique to improve system resilience, robustness, and security by deploying functionally-equivalent, yet different software versions on multiple systems [16]. In all such cases, updated software versions are required to be deployed only on a sub-set of the device fleet. Another common scenario is to limit the simultaneous use of some licensed software to a specific number of devices. Assignment in such circumstances will depend on whether other devices belonging to the same user are already running this software or not.

II. Multi-dimensional context awareness. In the context of rather static and homogeneous computing nodes (*e.g.* a computing cluster or a data centre), software management typically takes into consideration only the cyber-dimension of the target nodes, such as available hardware, networking configuration, OS version, already installed software, *etc.* As opposed to this, correctly assigning software components at the edge often needs to take into account a much richer and more diverse context of edge infrastructures [9]. To this end, we propose adopting a three-fold context model, consisting of the **cyber**, **physical**, and **social** dimensions, as explained below:

1. **Cyber** dimension covers the hardware and software aspects of edge gateways. For example, software assignment on a specific gateway may depend on connected sensors/actuators, OS version, security patches, networking interfaces (*e.g.* wireless/wired), power source (*e.g.* battery/constant power supply).
2. **Physical** dimension primarily refers to where exactly an edge gateway is deployed physically. As explained in Section 3, in this paper we consider connected

edge infrastructures deployed on end users' premises. For many scenarios (*e.g.* precision agriculture or environmental monitoring), this means that edge gateways may be exposed to changing and possibly harsh weather conditions, such as high/low temperatures or extreme humidity. Accordingly, vendors may opt for not assigning compute intensive AI-driven applications to those nodes already exposed to high temperatures not to cause overheating. Another example comes from telemedicine, where monitored patients are advised to take gateways with them even when travelling (*e.g.* in a camping trailer or in a cruise). In such cases, the updated physical placement or even the GPS location may be taken into account to assign a matching software update. The last but not the least, the physical placement is crucial for software assignment as far as security-related software management is concerned due to the direct affect on the current trustworthiness of the target device.
3. **Social** dimension captures the human-related aspects of edge gateways, which, as previously explained, are deployed on end users' premises and may be even equipped with some physical user interfaces for direct interaction. A good example of the social dimension is the subscription level, which defines the 'richness' of applications features a user might have. Coming back to the telemedicine example, only premium subscribers may be given a possibility to take their gateways on a trip, whereas non-premium subscribers will be blocked based on the updated GPS position. Another example from telemedicine is advanced AI-driven image recognition to be deployed directly on cameras for immediate fall detection or some other urgency.

Taken together, the three dimensions provide a useful conceptual framework for modelling heterogeneous device contexts. It is worth noting that these two design principles can be applied together to allow modelling situations where one device is part of a context of another device within a fleet. This way, we are able to capture the contextual inter-dependencies existing between the devices at the fleet level. For example, assigning a new software version to a device may depend on what other devices are connected and what software they are already running. We will discuss such relevant use cases in more detail in Section 5.

5 Proof of Concept

We now proceed with explaining how the proposed context modelling principles can be put in practice. For convenience, we will base our demonstration on a relevant example of remote patient monitoring (RPM), which we will explore through several use case scenarios related to day-to-day software maintenance.

5.1 Running Example: Edge Gateways for Remote Patient Monitoring

The use case scenario refers to a telemedicine provider, who offers RPM services. For each customer (typically – elderly people living at their own residences), they provide a healthcare gateway, *i.e.* a small single-board computer similar to Raspberry Pi, along with a set of medical sensors, cameras, and wearable emergency

beepers. Each gateway collects measurements from Bluetooth-connected downstream sensors (*e.g.* blood pressure, glucose and oxygen levels) or cameras (*e.g.* video stream for fall detection) and, before sending them to the back-end cloud service, may also perform some local (pre-)processing and aggregation. The patients and their caretakers have access to the data via a Web interface and a mobile app. Some patients also opt for battery-powered portable gateways to have a possibility to carry them with the essential sensors whenever they are away, *e.g.* relocating to a summer house, travelling in a camper trailer or on a cruise ship. During these periods, the gateway connectivity switches from the residential WiFi network to a mobile one (3G or 4G). The gateways also differ in terms of their hardware capacity – *i.e.* the older versions are less powerful than the modern ones in terms of CPU, RAM, and available disk storage. At present, the device fleet comprises more than 500 gateways installed on the customers' premises, and there also exist several versions of the edge RPM software. More specifically, there is a version dependent on additional disk storage to accumulate sensor measurements and occasionally send them in batches, whenever a device is not connected to a WiFi network (*e.g.* in a trailer camper). Another version requires increased CPU and RAM resources, since it not just transfers data, but also implements local data pre-processing. There is also a version enhanced with support for image processing for fall detection, and therefore requires several cameras to be installed and connected. Depending on the subscription level, some versions provide only basic limited features, while some other are richer in terms of offered functionality. All these examples demonstrate how the heterogeneous context determines what RPM software version has to be assigned and installed on each individual gateway in the fleet.

The development team continuously updates the edge software by adding new features and patching bugs, following agile practices. After each DevOps cycle, typically executed on a weekly basis, there is a mass re-deployment involving many or sometimes all the devices in the fleet. To perform the assignment, it is required to parameterise each deployment and match them to devices and their cyber-physical-social context properties.

5.2 Technological Baseline: Eclipse Ditto

As the technological baseline on top of which we have developed our proof of concept implementation, we have used Eclipse Ditto[8] – an open-source framework for building DTs of Internet-connected devices with extensible modelling and built-in querying languages. Ditto acts as middleware, providing an abstraction layer for IoT solutions interacting with physical devices via the DT pattern. It can be seen as a toolkit, providing some core functionality (*e.g.* meta-model, database, different messaging protocols and connectors, REST APIs, *etc.*), while some other features have to be written by users on top of them (*e.g.* domain-specific DT models, graphical user interfaces, device-side monitoring agents, *etc.*). Being part of a larger open-source ecosystem, it can be relatively easy

[8] `https://www.eclipse.org/ditto/`

integrated with some other technologies from the Eclipse stack, including various communication protocols, pub-sub messaging and load balancing.

Eclipse Ditto meta-model. Ditto offers developers a meta-model, which in its simplest form enables modelling physical entities (*i.e.* things) using a JSON schema with the following key concepts (as depicted in Fig. 3):

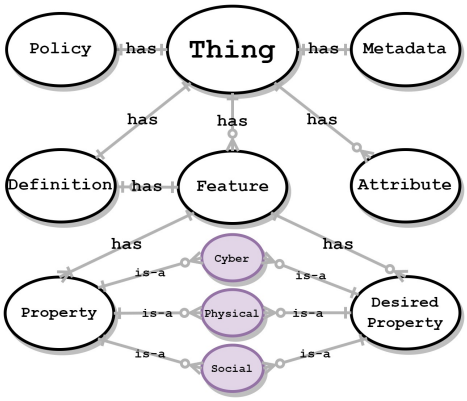

Fig. 3. Eclipse Ditto meta-model.

– `Thing` is the top-level modelling concept for describing physical assets.
– `Definition` is included in every `Thing` (and optionally in `Features`) and essentially represent a URI linking to an external WoT model. It describes how a `Thing` is structured and which behaviour/capabilities can be expected in a interoperable and standard manner.[9]
– `Policy` enables to configure fine-grained access control. A specific policy defines who and how can access a specific resource. Although very important for the overall security trustworthiness of the DT system, `Policies` are beyond the scope of this work.
– `Metadata` is Ditto's internal field to store technical information, *e.g.* version or creation/modification timestamps.
– `Attributes` are used to model rather static properties of a `Thing`, *i.e.* values that do not change as frequently as `Features`. They can be of any type and can be used to search for `Things`.
– `Features` are the central modelling concept to capture all run-time data and functionality of a `Thing` in a given application system. Users are allowed to define their own `Features` or extend existing WoT definitions. This is a key enabler for modelling the multi-dimensional context through more fine-grained `Properties`.
– `Properties` are used within `Features` to model individual run-time indicators of a `Thing`, *e.g.* to manage the status, the configuration or any fault information. Each `Property` can be either a simple scalar value or a complex JSON object. By using `Properties`, it is possible to implement the prominent *desired-reported* pattern widely adopted in DTs, wherein sensor measurements are reported upstream, while desired configuration updates are pushed downstream, until eventually `Properties` and `DesiredProperties` are in sync.

Extending the default Ditto functionality. The default Ditto functionality was extended in the following ways:[10]

[9] We further discuss this functionality in Section 6 as part of future work.
[10] `https://github.com/SINTEF-9012/ditto-fleet`

– Graphical user interface (based on the NodeJS/React stack) to offer the RPM provider the required application-specific software management features.
– Device-side monitoring agents for collecting contextual information and receiving updates.
– Back-end service for exchanging and synchronising desired (*i.e.* pushed by the RPM provider) and reported (*i.e.* collected by monitoring agents) properties.
– Extended DT definitions of RPM gateways including the cyber-physical-social context information (*i.e.* `Properties`) received from monitoring agents, as part of the application-specific fleet model. The gray circles in Fig. 3 represent these extensions to the `Property` concept.
– Software assignment logic based on Resource Query Language (RQL)[11] making use of the enriched DT definitions.

The latter two extensions are particularly relevant in the context of this paper. The simplified JSON snippet in Listing 1.1 demonstrates the overall structure of the DT. Please note the three blocks within `Features`, corresponding to cyber, physical and social context properties of the managed gateways. For clarity purposes, we omit extensive definitions and will focus on some practical use cases in the next subsection. In practice, the definition of a DT is expected to be much longer in order to accommodate all possible contextual information required for software assignment.

Listing 1.1. A sample of an RPM gateway digital twin.

```
{
  "thingId": "no.sintef.sct.giot:rpm_gateway_01",
  "policyId": "no.sintef.sct.giot:rpm_policy",
  "definition": "no.sintef.sct.giot:rpm_gateway:1.0.0",
  "attributes": {
       "manufacturer": "RPM Inc.",
       "cpu_model": "Broadcom BCM2711"
       "cpu_arch": "arm_v8",
       "os": {
           "name": "Raspberry Pi OS 11 (Bullseye)",
           "kernel_version": "5.15.84"
       }
   },
   "features": {
       "cyber": {
           "properties": {
               "monitoring_agent": {
                   "enabled": true,
                   "version": "1.0.0"
               },
               "docker_engine": {
                   "enabled": true,
                   "version": "containerd.io_1.2.0-1_arm64.deb"
               },
               "proxy": {
                   "enabled": false,
                   "host": "",
                   "port": ""
               },
               "ssh": {
                   "enabled": false,
                   "port": 443
               },
```

[11] `https://github.com/persvr/rql`

```
                "dev_env": "testing",
                "fingerprint_sensor": true,
                "2fa": false
            }
        },
        "physical": {
            "properties": {
                "deployment_site": "hospital",
                "gps_location": {
                    "latitude": "59.945283167835846",
                    "longitude": "10.713121330182533"
                }
            }
        },
        "social": {
            "properties": {
                "user_subscription": {
                    "type": "premium",
                    "expires": "2023-12-31T23:59:59Z"
                }
            }
        }
    }
```

5.3 Use Case Scenarios

Ditto comes with an RQL-based query language to search for devices within the managed fleet using their attributes and features and a collection of logical and arithmetical operators. We now present a series of common use cases scenarios serving to demonstrate the use of context-aware DTs in day-to-day software maintenance activities of the RPM provider. The examples use RQL to search for target devices within the fleet and use conventional logical operators. Please note that the code snippets with queries are somewhat simplified for demonstration and clarity purposes.

Use case 1: targeted maintenance of a single device. A very simple, yet common task is to perform remote maintenance on a single device, *e.g.* during live interaction with a customer over the phone. In such cases, the device ID is known, and some test commands can be executed using this query.

```
eq(thingId,"no.sintef.sct.giot:rpm_gateway_01")
```

Use case 2: maintenance of devices based on a single context property. Another common scenario is to target a specific subset of devices based on some single attribute or feature. It can be, for example, a new version release for all ARM-based gateways:

```
eq(attributes/cpu_arch,"arm_v8")
```

Similarly, it is also a common practice to release some experimental features to a testing environment, before pushing it to production:

```
eq(features/cyber/properties/dev_env,"testing")
```

To a great extent, the first two use case scenarios represent the current state of practice as far as DT-based software assignment is concerned.

Use case 3: maintenance of devices based on a multi-dimensional context. This is a more advanced scenario, which needs to take into account several features to release agile updates only to those devices falling into the target scope of the update command. Depending on the strictness of the target conditions, it can be none, a few or all devices from the fleet. The following query refers to a situation when a new software version with mandatory fingerprint-based authentication needs to be pushed to a device equipped with such a sensor (**cyber**) and where the two-factor authentication is not yet enabled (**social**).

```
and(eq(features/cyber/properties/fingerprint_sensor,"true"),
  eq(features/social/properties/2fa, "false"))
```

Another example is releasing new AI-driven wellness recommendation features only to GPU-enabled devices (**cyber**), do not get overheated (**physical**), and have a premium subscription (**social**). For simplicity, we define 'overheating' as an exceeding of the mean value of recent CPU temperature observations (40°).

```
and(eq(features/cyber/properties/gpu,"true"),lt(mean(features/physical/
    properties/cpu_temp),"40"),eq(features/social/properties/user,"premium"))
```

Use case 4: releasing an experimental feature to a limited number of devices This scenario refers to a situation when new software features should be safely tested only on a very small number of devices in a production environment. For example, the assignment logic can target a single device within a larger installation of 10+ gateways in the same medical institution. Accordingly, this combined query will first check if there are more than 10 gateways installed, and, if yes, will return a single device among the available. This is a demonstration of an assignment logic at the fleet level, wherein the decision is based taking into account other neighbour devices.

```
and(first(eq(features/physical/properties/deployment_site,"hospital")),
  gt(count(eq(features/physical/properties/deployment_site,"hospital")),10))
```

Use case 5: evenly distributing 2 software versions within the fleet The next related example refers to the software diversification strategy, often applied to increase the overall security of the fleet. Briefly, the idea is to run multiple software versions (functionally identical, yet different at the code level). This can be achieved by first querying the total number of target gateways (*e.g.* running a specific Raspbian OS), and then equally splitting them into two.

```
num = count(eq(attributes/os/kernel_version, "5.15.*"))              (1)
limit(count(eq(attributes/os/kernel_version, "5.15.*"), 0, num/2)    (2)
```

5.4 Assumptions and Limitations

The described query snippets serve to demonstrate the feasibility of the proposed approach. Admittedly, in practice they will be somewhat more complex and expressive, as well as the definition of the DT in Listing 1.1. Despite the fact that current proof of concept is based on Eclipse Ditto and its internal modelling

and querying languages, the high-level design concepts can be applied to and implemented in other DT platforms allowing extending the definition of DTs with cyber-physical-social context properties.

An important assumption to make, however, is the need to design and implement own monitoring agents. These agents will be deployed on devices to collect run-time multi-dimensional metrics about the managed devices, to be collected by the centralised DT platform. In many occasions, these monitoring agents would also need to have some elevated access rights to be able to probe low-level information about hardware or host OS. Admittedly, as with many modelling approaches, there is usually more than one way of representing the surrounding world with digital models. While the proposed design principles can provide a high-level modelling framework, some extensive modelling is still needed to make the DTs usable in practice. To this end, some deep knowledge of the target application system is required.

6 Conclusion and Future Work

In this paper, we answer the research question posed in Section 1 and aimed to address the challenge of limited context-awareness in the existing DT platforms – an important pre-requisite to perform assignment for software maintenance in edge application systems. We brought forward the two design principles required for modelling DTs. First, it is important to focus on the overall fleet, rather than on individual devices. This way, it is possible to capture and evaluate possible inter-dependencies between the devices, as well as global goals of the overall edge system. Second, it is required to go beyond the traditional hardware and software context properties, but also take into consideration the physical and social dimensions of edge devices, which are often installed on end users' premises and provide an interface for physical interaction. By enabling such multi-dimensional cyber-physical-social context awareness, we provide a much richer foundation for performing software assignment across a wide range of business scenarios. The latter is demonstrated in the context of an RPM edge application, using the existing DT framework Eclipse Ditto. In this proof of concept implementation, we we were able to model the device fleet following the two design principles, which supported a series of software maintenance use case. Although the overall results are positive, the proposed solution to a certain degree still requires manual modelling effort, since proposed design principle are too high-level, while the described prototype implementation is specific to the RPM scenario and underlying Ditto implementation. Another related limitation is the need for device-specific monitoring agents, which will collect multi-dimensional context information required for DTs. Nevertheless, this is a work-in-progress research effort, which we ar eplanning to further improve in the following directions:

1. **Integration with WoT Thing Description and semantic modelling**: WoT Thing Description is used to model the metadata and interfaces of physical things to enable integration of heterogeneous devices and interoperability

across diverse applications. They are encoded in a JSON format that also allows JSON-LD processing. The latter provides a promising foundation to represent knowledge managed devices in a machine-readable way. Even further possibilities for automated reasoning can be unleashed with the adoption of Semantic Web ontologies, which are based on Description Logics and come with multiple automated reasoners and modelling editors. Ditto can support both technologies, meaning that assignment may be implemented using more expressive tools, going beyond the the simple RQL reported in this paper.
2. **Using a graph-based database**: As we advanced with the experiments on Eclipse Ditto, it became apparent that entities and relationships describing the multi-dimensional context of devices within a fleet do not always fit into the fixed nested structure of JSON documents, adopted by Eclipse Ditto and its underlying document-oriented MongoDB database. A promising research direction is to adopt a *graph-based* representation of a managed fleet. Graph-based DTs is a prominent paradigm, mainly due to a more intuitive representation of data, which will then be easier to query for a human by traversing the graph elements. Arguably, a graph is a more natural abstraction for a device fleet, with all the inter-dependencies and heterogeneous contexts of individual devices.
3. **Empirical evaluation**: Even though we have used the running RPM example to demonstrate the viability of the proposed approach, it still needs to undergo a proper empirical validation by DevOps engineers to prove its applicability in enterprise-level production environment. This will be implemented as part of the ongoing work in the R&D projects acknowledged below.

Acknowledgements This work is co-funded by the European Commission's HEU and H2020 Programmes under grant agreements 101070455 (DYNABIC), 101095634 (ENTRUST) and 101020416 (ERATOSTHENES), and the Research Council of Norway's BIA-IPN programme under grant agreement 309700 (FLEET).

References

1. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Model Checking, pp. 305–343. Springer (2018)
2. Bellendorf, J., Mann, Z.Á.: Classification of optimization problems in fog computing. Future Generation Computer Systems **107**, 158–176 (2020)
3. Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: Satisfiability modulo theories and assignments. In: International Conference on Automated Deduction. pp. 42–59. Springer (2017)
4. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog: State of the art and open challenges. Software: Practice and Experience **50**(5), 719–740 (2020)
5. Brown, P., Burleson, W., Lamming, M., Rahlff, O.W., Romano, G., Scholtz, J., Snowdon, D.: Context-awareness: some compelling applications. In: Proceedings the CH12000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness (2000)
6. Charpenay, V., Käbisch, S.: On Modeling the Physical World as a Collection of Things: The W3C Thing Description Ontology. In: Proceedings of the 17th International Semantic Web Conference (ESWC 2020). pp. 599–615. Springer (2020)

7. Da Silva, D.M.A., Sofia, R.C.: A discussion on context-awareness to better support the iot cloud/edge continuum. IEEE Access **8**, 193686–193694 (2020)
8. Dautov, R., Distefano, S.: Stream processing on clustered edge devices. IEEE Transactions on Cloud Computing **10**(2), 885–898 (2020)
9. Dautov, R., Distefano, S., Bruneo, D., Longo, F., Merlino, G., Puliafito, A.: Data Processing in Cyber-Physical-Social Systems Through Edge Computing. IEEE Access **6**, 29822–29835 (2018)
10. Dautov, R., Distefano, S., Buyya, R.: Hierarchical data fusion for smart healthcare. Journal of Big Data **6**(1), 1–23 (2019)
11. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An ontology-based context model in intelligent environments. arXiv preprint arXiv:2003.05055 (2020)
12. Gubert, L.C., da Costa, C.A., Righi, R.d.R.: Context awareness in healthcare: a systematic literature review. Universal Access in the Information Society **19**, 245–259 (2020)
13. Guerrero, C., Lera, I., Juiz, C.: Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures. Future Generation Computer Systems **97**, 131–144 (2019)
14. Jacoby, M., Usländer, T.: Digital twin and internet of things—current standards landscape. Applied Sciences **10**(18), 6519 (2020)
15. Kaenampornpan, M., O'Neill, E., Ay, B.B.: An integrated context model: Bringing activity to context. In: Proc. Workshop on Advanced Context Modelling, Reasoning and Management (2004)
16. Larsen, P., Brunthaler, S., Franz, M.: Security through diversity: Are we there yet? IEEE Security & Privacy **12**(2), 28–35 (2013)
17. Leivadeas, A., Kesidis, G., Ibnkahla, M., Lambadaris, I.: VNF placement optimization at the edge and cloud. Future Internet **11**(3), 69 (2019)
18. Magdich, R., Jemal, H., Ben Ayed, M.: Context-awareness trust management model for trustworthy communications in the social internet of things. Neural Computing and Applications pp. 1–26 (2022)
19. Merlino, G., Dautov, R., Distefano, S., Bruneo, D.: Enabling workload engineering in edge, fog, and cloud computing through openstack-based middleware. ACM Transactions on Internet Technology (TOIT) **19**(2), 1–22 (2019)
20. Minerva, R., Lee, G.M., Crespi, N.: Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. Proceedings of the IEEE **108**(10), 1785–1824 (2020)
21. Qian, C., Liu, X., Ripley, C., Qian, M., Liang, F., Yu, W.: Digital twin—cyber replica of physical things: Architecture, applications and future research directions. Future Internet **14**(2), 64 (2022)
22. Semeraro, C., Lezoche, M., Panetto, H., Dassisti, M.: Digital twin paradigm: A systematic literature review. Computers in Industry **130**, 103469 (2021)
23. Song, H., Dautov, R., Ferry, N., Solberg, A., Fleurey, F.: Model-based fleet deployment of edge computing applications. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. pp. 132–142 (2020)
24. Song, H., Dautov, R., Ferry, N., Solberg, A., Fleurey, F.: Model-based fleet deployment in the iot–edge–cloud continuum. Software and Systems Modeling **21**(5), 1931–1956 (2022)
25. Xu, D., Li, Y., Chen, X., Li, J., Hui, P., Chen, S., Crowcroft, J.: A survey of opportunistic offloading. IEEE Communications Surveys & Tutorials **20**(3), 2198–2236 (2018)