Advanced Engineering Informatics 33 (2017) 243–257

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Platform as a service gateway for the Fog of Things

Nandor Verba [a],*, Kuo-Ming Chao [a], Anne James [a], Daniel Goldsmith [a], Xiang Fei [a], Sergiu-Dan Stan [b]

[a] Faculty of Engineering, Environment and Computing, Coventry University, Coventry. United Kingdom
[b] Faculty of Mechanical Engineering, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

ARTICLE INFO

Article history:
Received 1 July 2016
Received in revised form 8 November 2016
Accepted 21 November 2016
Available online 30 November 2016

Keywords:
Internet of Things
Gateway
Fog Computing
Horizontal integration
MQTT
OSGI
Cloud manufacturing

ABSTRACT

Internet of Things (IoT), one of the key research topics in recent years, together with concepts from Fog Computing, brings rapid advancements in Smart City, Monitoring Systems, industrial control, transportation and other fields. These applications require a reconfigurable sensor architecture that can span multiple scenarios, devices and use cases that allow storage, networking and computational resources to be efficiently used on the edge of the network. There are a number of platforms and gateway architectures that have been proposed to manage these components and enable application deployment. These approaches lack horizontal integration between multiple providers as well as higher order functionalities like load balancing and clustering. This is partly due to the strongly coupled nature of the deployed applications, a lack of abstraction of device communication layers as well as a lock-in for communication protocols. This limitation is a major obstacle for the development of a protocol agnostic application environment that allows for single application to be migrated and to work with multiple peripheral devices with varying protocols from different local gateways. This research looks at existing platforms and their shortcomings as well as proposes a messaging based modular gateway platform that enables clustering of gateways and the abstraction of peripheral communication protocol details. These novelties allow applications to send and receive messages regardless of their deployment location and destination device protocol, creating a more uniform development environment. Furthermore, it results in a more streamlined application development and testing while providing more efficient use of the gateway's resources. Our evaluation of a prototype for the system shows the need for the migration of resources and the QoS advantages of such a system. The examined use case scenarios show that clustering proves to be an advantage in certain use cases as well as presenting the deployment of a larger testing and control environment through the platform.

1. Introduction

The concepts from the Internet of Things (IoT) are of key interest to researchers and industry leaders [1]. New initiatives like Germany's Industry 4.0 [2], as well as concepts from [3] consider the interconnection of devices as the fourth industrial revolution which is estimated to result in over 21 billion devices connected to the Internet by 2020 [4]. Such systems would require homogeneous and interoperable Machine to Machine (M2M) networks that can be accessed from the Internet while abstracting unwanted details and enabling higher level application development and better use of resources.

There are multiple problems and directions that can be taken to create a fully Automated Manufacturing Environment. The IoT oriented components focus on the orchestration of resources that are needed to make products, reducing time to market, manufacturing times and idle devices and resources. Previous research has proposed various solutions. In [5] model-based task and deployment method is suggested while in [6] a Service Oriented Manufacturing (SOM) solution is presented for mapping and access. Furthermore, the platforms presented in [7,8] suggests the collaboration of a number of higher-level systems to meet these requirements.

When discussing the IoT we can consider three distinct approaches for the architectures of such systems [9]. An IPv6 based network where devices are uniquely accessible through Constrained Application Protocol (CoAP) or other lightweight protocols has been suggested in [10], while in the cloud oriented approach devices are accessed through API's [11] or using Message Queue Telemetry Transport (MQTT) protocol. The middleware approach is based on gateways or brokers that communicate with devices

* Corresponding author.
E-mail addresses: verban@coventry.ac.uk (N. Verba), csx240@coventry.ac.uk (K.-M. Chao), csx118@coventry.ac.uk (A. James), aa9863@coventry.ac.uk (D. Goldsmith), aa5861@coventry.ac.uk (X. Fei), sergiu.stan@mdm.utcluj.ro (S.-D. Stan).

http://dx.doi.org/10.1016/j.aei.2016.11.003
1474-0346/© 2016 Elsevier Ltd. All rights reserved.

through more lightweight communication protocols such as 6LoWPAN, nRF24L01 or ZigBee and forward these messages to the cloud or other clients such as in [12].

With the introduction of Cloud and Fog Computing paradigms, the use of resources available at the edge of the network is considered as well as the deployment of application and processing tasks on the edge devices [13]. Proposals like MADCAT in [14,15] suggest large applications be decomposed into components and deployed onto devices while [16] suggests a MapReduce like approach with IoT application development. Together with proposals from [17] which looks at reconfigurable components and [18] which looks at agent based cooperative smart objects, these suggest a need for Software Defined Networking as well as the need of decomposing applications into components and running them on the gateway.

The idea for the use of resources on the edge of the network was first introduced by Cisco [19]. Advances in Networking as a Service (NaaS) and the increased processing power of gateway devices have led to the development of edge computing platforms like Docker [20]. Edge computing includes solutions based on Virtual Machines [21,22] as well as container based application deployment solutions.

Open Service Gateway Interface (OSGI) is a modular service platform for Java that has been the focus of research towards modular IoT Gateways. OSGI can be used for multi-tenant cloud connection architecture as in [23]. Platforms like HEPA [24] propose the use of Zookeeper to control a set of OSGI Gateways that would facilitate the transmission and translation of device information. One of the drawbacks of the OSGI core platform is that it lacks solutions for asynchronous communication between components. To address this issue [25,26] proposed an a messaging based solution that maps messages to either internal services or to an event administration system component.

The requirements presented in [2] for the industrial use of IoT Devices as well as innovations in supply chain management presented in [27] show that the existing gateways and solutions require extensive research on the horizontal integration of devices and gateways to allow for multiple devices to send and receive messages between each other regardless of the underlying protocol. Existing gateways focus on a more vertical approach where systems are built upon one single platform or language, which results in system lock-in and reduces the number of use cases and overall functionality of the gateway. Vertical integration is concerned with integrating devices and protocols from a single provider and expanding their capabilities and scope, while horizontal integration focuses on providing inter-platform support and cross-communications solutions to users so that resources from various suppliers can interoperate.

When discussing message passing the scenarios that need to be considered are Machine to Machine (M2M), Machine to Gateway (M2G), Machine to Cloud (M2C) and their combinations as described in [28]. The translation and forwarding of these messages is difficult, due to the existing protocol fragmentation in the industry as well as due to each protocol having their own advantages and specific use cases. There is a need for translation of messages from different protocols as suggested in [29] but the bigger issue is to allow for applications to be deployed in a protocol agnostic environment where all the layers of communication are abstracted away from applications. This is a feature that is missing from existing gateway architectures that merely provide means to directly address physical resources but fail to provide higher level abstraction of device communication.

The exploitation of smart gateways for Wireless Sensor Network (WSN) use cases such as smart-office and wide area monitoring requires that devices, and their events and controls be available from a wide variety of locations to allow for more complex applications to be deployed without the need to redesign them. This can only be done if we consider a local cluster of gateways that share information and devices, which allows for redundancy, load balancing and high availability. Current gateways allow for a horizontal implementation of similar features that would enable applications components to communicate with each other. While this solution is sufficient for certain implementations, they lack an implicit implementation of clustering which would allow messages and events to be passed without the need to program migrations and the deployment of applications to multiple gateways.

We propose a messaging oriented gateway architecture that allows for multiple application containers and drivers to be deployed and connected on the same device while enabling application migration. This architecture would allow multiple connections to different providers with different privileges as well as a regional clustering and the use of local resources such as storage and location services.

This gateway architecture would allow the horizontal integration of multiple platform devices through the connection of the messaging service and the abstraction of protocol specific information. Messages would be passed from multiple protocols and drivers from different gateways to one application deployed to one gateway, reducing the impact of protocol fragmentation as well as creating a coherent application environment inside the cluster allowing for message routing from one application to another without these having to be configured for regional communication. These additional functionalities reduce the complexity of singular applications and allow larger systems to be deployed without individual gateway (re)configuration. Such an environment is needed in order to be able to deploy applications to devices without platform lock-in or other prohibitive factors.

The rest of the paper is organized into a state of the art chapter, a chapter dedicated to describing the proposed architecture, one for implementation and performance evaluation and a conclusion and future work chapter. Section 2 evaluates the state of the art and investigates the requirements of horizontal integration of gateways. Section 3 presents the proposed architecture, showing how it meets the requirements identified in Section 2, as well as explaining its construction. The final two sections look at evaluating the performance of the Gateway, together with use case scenarios that highlight the added functionalities, followed by the conclusions drawn from these as well as future work.

2. State of the art

IoT gateways have become increasingly configurable and their functionalities have expanded. The horizontal integration directives aim at allowing platforms and devices from different providers using different protocols to interact. This would increase reusability and reduce application complexity. To allow the connection of multiple devices we need multi M2M protocol support, as well as registration, management and an enhanced configurability for these devices. The increased number of resources available on the gateway has led to the need to be able to virtualize these and move a portion of the resource use from the cloud to the edge of the network.

There are a number of different approaches to the design and implementation of IoT gateways. Most of the initial approaches as well as some of the latest ones like Eclipse Scada, Krikkit, SmartHome and HePA [24] concentrate on semantic interpretation of data and configuration based routing or event creation. Other approaches like that of Kura and Eliot look at fully reconfigurable systems where applications configure and define everything, a fully modular system. These approaches cause platform and provider lock-in where information passing between peers is problematic. Solutions like BUTLER [30] and the use of eTrice provide an abstraction of protocols to enable an easier application development. While eTrice [31] generates Java or C code based on the written code, Butler deploys the runtime environment directly onto devices, which allows the user to review the written code, rather than the generated one for errors and library issues. Most of the presented gateways have very limited solutions to certain aspects of gateways like reconfiguring and reprogramming connected devices to suit the needs of the users. Proposals like GITAR [17] provide a platform that can reconfigure embedded devices connected to it. In summary, existing platforms all focus on specific aspects of IoT gateways, allowing certain aspects to be configured or reprogrammed and some also allowing for applications to be deployed on them. A functionality based summary review can be seen in Table 1 where the differences between the gateway platforms are highlighted based on 6 criteria.

Due to the differences in processing and storage resources of IoT devices there is a wide range of tailored M2M protocols. The higher level protocols like CoAP, SNMP and MQTT-SN are used on devices that have higher processing and power resources available. More resource constrained devices use protocols with lower levels of abstraction and functionality, such as 6LoWPAN, XBee, RF24 or even core 434 MHz, each having their preferred implementation scenarios and varying advantages and disadvantages. This protocol fragmentation has led to increased research regarding the brokering and semantic translation of received messaged from the existing protocols such as in [29]. Architectures like Krikkit and BUTLER look at mapping to REST requests with notification feedback. In contrast, BUTLER attempts the handling of asynchronous requirements of IoT Systems with the use of a Messaging service that provides this implicitly. The solutions like Kura and ELIoT use MQTT as a messaging service with the cloud and translate all device messages to MQTT. The drawback of most of the presented solutions is that while they offer a uniform and configurable communication means with the cloud, they do not provide a protocol agnostic message passing system for applications and device messaging.

The management and northbound or cloud oriented connections of the gateways have a number of approaches that can be used. Gateways like Krikkit, Eclipse SCADA, Kura, SmartHome and BUTLER all use RESTful APIs and User Interfaces to control and manage them. Platforms like Kura and Krikkit allow for MQTT cloud connections to be configured for message passing. BUTLER allows for multiple cloud tenancies through connections made through the REST APIs which can connect to local area and cloud resources as well as other smart devices. Approaches like HePA suggest proxying through CoAP for passing of control and device data between gateways. While these approaches allow for some basic networking configuration they lack a truly software defined networking platform that would support the configuring of multiple networking connections that not only allow message passing but management and application deployment as well.

The requirements for the horizontal integration of devices has become evident in the past years with an increased amount of platforms switching from a vertical view, where platforms have their specific protocol and device support, to a more horizontal one, encapsulating different protocols and device connections from other providers. This is leading to an increased interconnectivity and the use of multi tenancy connections for features available from different providers. Most of the presented platforms like Kura only allow one MQTT connection to be configured. While applications can implement the drivers and have their own connection, this is not implicit to the platform. BUTLER provides the most extensive support for this, supporting multiples types of devices like smart phones, local computers, gateways and cloud connections. In general however, these platforms provide a mostly vertical view of the system with connected devices and messages still being confined to their respective gateways and these needing to be updated and deployed independently. There is a requirement for a more loosely coupled connection among devices, applications and resources, where these applications can pass messages seamlessly from different containers to devices connected to other available gateways without needing to be rewritten. This would allow for functionalities like migration, clustering and high availability to be explored for these gateways which could lead to higher quality of service (QoS) standards.

The use of resources at the edge of the network is one of the main concerns of Fog Computing as described by Cisco [19], with the use of processing and networking resources being of main concern. Some platforms suggest deploying VMs such as in ELIoT or allow the users to configure the data processing as with the Krikkit, SCADA and SmartHome platforms. Solutions like Kura, eTrice and BUTLER suggest the deployment of applications onto these gateways which allows for faster deployment and more efficient use of resources but constrains the users to platform or language whereas VMs allow for full control of the environment. Although the storage resources available on the gateways are rarely discussed, there is research where the use of software like CouchDB and PouchDB for Fog computing devices is evaluated [32]. In these scenarios, device related information is stored locally and updated with the cloud when needed. Context resources are made available to applications which may include location as in Kura or region information as made available in HePA. A more comprehensive view on how to manage these resources is needed as well as a need to be able to combine resources management systems from different languages and platforms.

Based on our review of the existing platforms as well as the direction of the IoT community, we can conclude that there is a need for more horizontal integration of gateways as well as a need for a protocol agnostic messaging system for applications to talk to

Table 1
Gateway platforms evaluation.

Gateway platform	Horizontal integration directives	Multi M2M protocol support	Multi cloud tenancy	Deployable application layer	Protocol agnostic messaging	Local resource use[a]
Krikkit – Eclipse	✓	✓			✓	R
Eclipse SCADA		✓				R, S
Kura – Eclipse		✓		✓	✓	P, N, S, O
Eclipse SmartHome		✓				P, R
eTrice	✓	✓	✓	✓		P, R
HePA	✓	✓	✓		✓	P, N
BUTLER	✓	✓	✓	✓	✓	P, N
GITAR	✓			✓	✓	P, N
ELIoT			✓		✓	P, N, S

[a] P - Processing, R - Message Routing, N - Full Networking, S - Storage, O - Other.

devices. Furthermore, in evaluating the current platforms we have seen that certain aspects have received a lot of attention and have had good solutions, especially in the case of BUTLER, but there is still a room for improvement. The resource availability and use by gateway applications, as well as in the creation of protocol agnostic and event based device messaging environment for the applications are key issues for the development of future platforms. In addition to the platform lock-in created by a vertical, single platform approach for most of the existing systems, device and protocol dependent solutions create a big impediment for application development for devices from multiple providers and protocols that provide similar functionalities, reducing the capabilities and reusability of such systems.

If gateways are to cope with the proposed interconnectivity and the wide range of devices, use cases, protocols and QoS requirements of future environments, they need to be able to connect to multiple cloud providers that may offer different data processing, storage and metadata analysis tools and features. Furthermore, these gateways need to allow for migration and clustering while maintaining device communication and application persistence within the cluster and the fog. Current approaches fail to provide an application environment to decouple deployment and messaging which is partly due to a lack of M2M protocol abstraction. They also fail to provide a virtualization layer for applications that allows complex application deployment using the resources of a set of gateways rather than the limited resources from single gateway. This becomes a particularly big issue for use cases where a highly interconnected and constantly reconfiguring environment is in place such as in the case of Smart Office and Home scenarios as well as Industrial Monitoring and Control applications that involve task and project based reconfiguring of the system.

3. Proposed gateway architecture

3.1. General view of platform requirements

The proposed Platform as a Service generic gateway architecture attempts to answer the requirements of an ever evolving IoT environment while improving on existing proposals especially on the topic of migration, clustering, abstraction and routing of device messages to the appropriate regions. The use of the resources available on the Gateway has been expanded from those suggested in [33] with the introduction of context information such as region, network information and location information. The review showed a need for a generic architecture that can encapsulate a wide variety of containers and drivers from different providers and languages.

This architecture is designed to fulfill the following requirements:

(1) *Protocol agnostic device messaging:* The messaging between devices and the application environment through the drivers is designed to allow for messages to be transmitted regardless of the devices' protocols or technologies. This allows applications to be oblivious to the underlying protocols or technologies with which they want to communicate. Furthermore, due to the routing of messages, applications can communicate with the devices from the cloud, or with the ones that are registered to other gateways on the local cluster.

(2) *Regional connections and messaging:* When gateways are deployed onto a WAN network they can form a local region which should allow information and messages to be shared between peers. This allows for faster message passing among local devices and with this connection clustering and high availability are also possible.

(3) *Multi-cloud tenancy:* The gateway should enable multiple cloud connections to be established in order for application and management information to be sent and received from these tenants.

(4) *Modular application deployment:* The application container should allow multiple applications to be deployed on the same gateway and communicate with each other so that complex applications can be deployed across simple components.

(5) *Application migration, clustering and testing functionalities:* Due to the nature of the gateway it needs to meet QoS requirements associated with the applications or cloud that it interacts with. These applications need to be tested and migrated seamlessly while maintaining inter-application and device communication in a secure environment.

3.2. Generic gateway architecture

This article introduces the notion of the Fog of Things as a Fog Computing platform that treats things as resources of the edge device and allows for a unified view and messaging with these devices. Fig. 1 shows the overview of the platform and the connections between components. The proposed gateway architecture is built around a new asynchronous messaging based model that allows the abstraction of different drivers and components by allowing messages to be routed to their destinations dynamically, based on a new header oriented routing model.

The proposed architecture offers a novel gateway design by increasing the horizontal integration of the gateways by allowing applications to send and receive information to and from a number of cloud providers using the configurable brokers. It also offers a wider range of client connection possibilities by providing WAN client connectivity through the configured regional connections. Another novelty presented by the gateway is the protocol agnostic container environment that allows applications to communicate with cloud providers, regional clients, peer applications, devices and requests resources through a unified medium without considering the underlying protocol for device, region or cloud communication. This is achieved through a set of brokers and drivers that translate and route these requests into messages understood by the respective sinks. The final novelty of the gateway is the possibility to configure WAN clusters of peer devices and migrate applications without the need of reprogramming them between the peers and available cloud containers.

This architecture and the associated components described in the following paragraphs can satisfy the requirements presented in the section above.

The gateway controller analyzes and deploys applications, as well as sending usage, load, capacity, connected device and region information to cloud and region clients. The gateway manages the non-admin tenant connections and the device drivers, and controls the regional authentication and registry. Finally, the gateway is capable of searching for available gateways in its WAN network. It can either enrol them to a region or create one and become its coordinator, if no peers are found. The controller manages the information about the capabilities of the gateway, its resources, the connected drivers and the available regional devices.

The Application Container is controlled and monitored by the Gateway Controller. Rather than having applications connecting to the Messaging Service directly, the Application container translates messages and events into its internal equivalents that can be understood by the deployed applications. This allows more applications to listen to the same broadcasted message, communicate between each other, and send information to the outside components asynchronously. Furthermore, this allows policies to be put on the devices like an internal firewall that would allow apps to

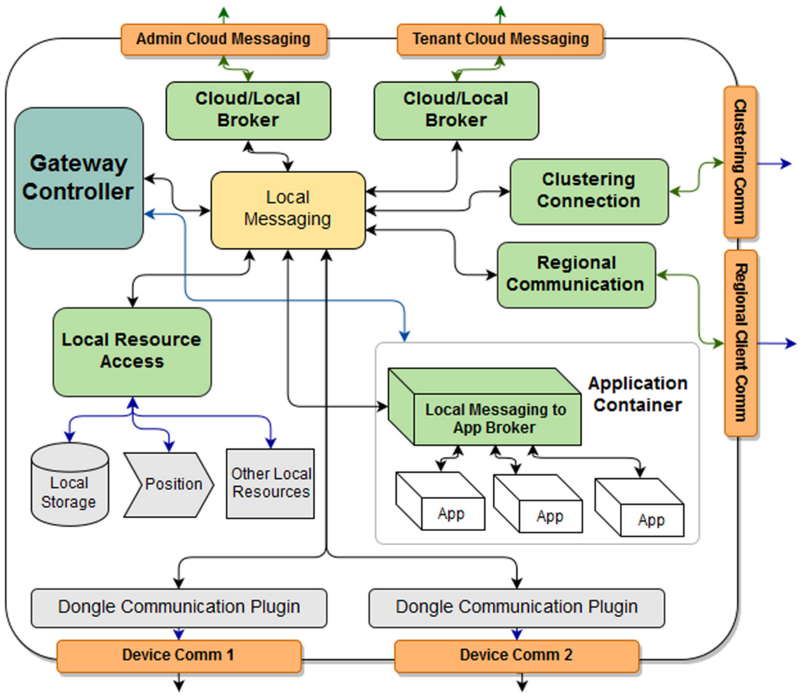

Fig. 1. Architecture of the gateway.

send and receive messages only from authorized or authentic sources.

M2M communication is fulfilled by the device communication components that are directly linked to the transceiver hardware and are also tasked with registering, authenticating and monitoring the devices. The received device messages are interpreted and sent to the corresponding sink through the messaging service, while messages sent from applications are encoded into the desired format and sent to the devices.

Cloud communication takes place through dedicated brokers that take messages directed at them, parse the headers and payload to the desired format and send them through the broker's medium, doing the reverse for received messages. This allows for different protocols to be used by tenant clouds to access applications and the gateway controller.

Storage and metadata information like location, regional clients, network information and other gateway details are considered local resources to the applications. Applications and devices are allowed to save data into databases, request location data and send the data to the application layer or the cloud.

The regional communication refers to two distinct communication methods. The first looks at gateways that can be discovered through a local network and that can be linked through the federation of the messaging service. The second method proposes the creation of regional access points to applications which can receive messages from a more varying range of local clients.

The details of the proposed components are described in detail in the following subsection. Each section looks at a major component of the framework and describes its functionality and proposed mechanism.

3.2.1. Local messaging service

The local messaging service is responsible for routing messages to the appropriate queue based on their headers and routing information. It is designed to support asynchronous messaging between components. Furthermore, new drivers and different configurations can be added to the gateway without modifying any applications or other components. In order to accomplish this, the messaging service is designed with a complex array of exchanges, which can be seen in Fig. 2.

The routing is designed in such a way that components can send messages in a generic format and the exchanges can route these messages based on the routing table on the gateway. The exchanges that routes messages to other components hold the group name of components (resources, devices, region, cloud, apps) and are designed to route the collected messages to the corresponding resolver components.

The message passing is designed for scaling, in order to support the addition of new components seamlessly and removal of old ones. Resolver exchanges allow messages to be routed to their specific queues based on header information and are the main configurable components to support the routing table in the messaging service.

Components are designed to communicate with other components by publishing messages to their specific exchange and retrieving messages from the queue in a unified way, without knowledge of the number or type of destinations of the message. This takes away the burden of reconfiguring the components when modifications on sources or destinations take place.

The control component is a special one, as it does not communicate with any other components through the messaging system but configures them on deployment, with the exception of the cloud connections which it uses to send and receive information and control parameters. The region component is connected to the container and cloud component, which is done in order to be able to route messages to applications which are deployed locally and to those which are deployed to the cloud.

3.2.2. Cloud controller and local resources

The cloud controller is responsible for configuring and deploying all the communication drivers with the cloud or the devices as well as managing the regional connections and authentication while relaying status information to the cloud.

The gateway sends status information to the specific cloud component by responding to requests that were made through the cloud connections. The first and main cloud connection has

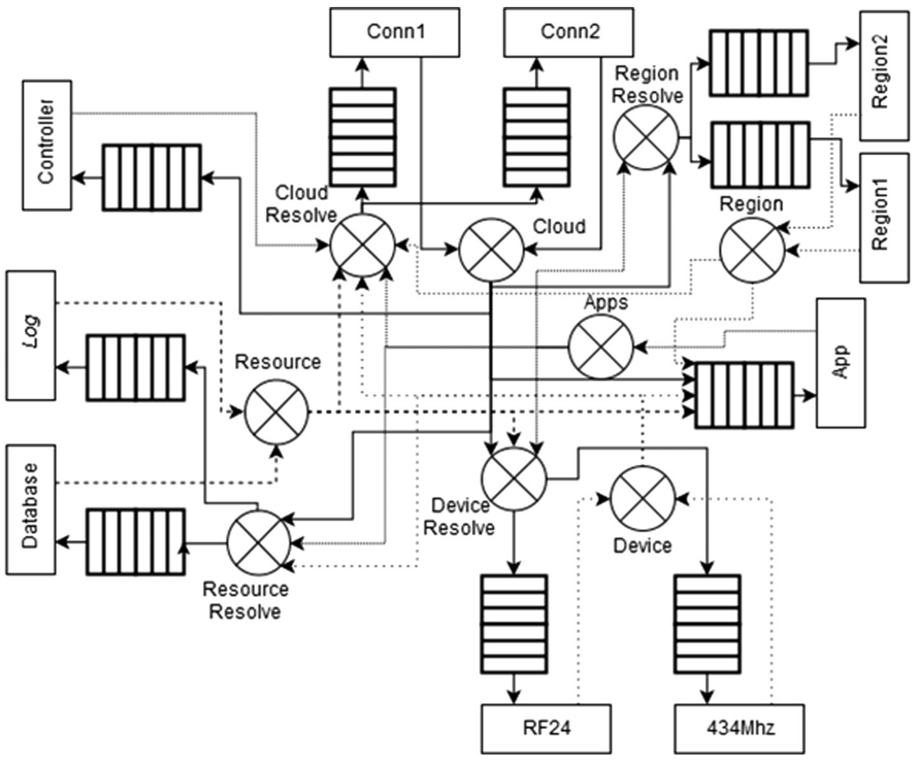

Fig. 2. Messaging exchanges and routing.

the most control over the system, as it is able to add and delete other connections, remove and modify apps deployed by other tenants as well as to set up the region communication and the device drivers. The other tenants are limited to offering and requesting authentication information for devices or regional agents as well as deploying and configuring their own applications and devices.

Local resources are controlled by the gateway controller and receive requests, data and commands through their respective drivers connecting them to the local messaging service and through this the applications. These local resources may include context information such as region parameters, location information, and storage. The storage component is a special one, because in contrast with other resources, it can contain metadata to support data requests. The resource can be configured for high-availability throughout the gateways as well as through cloud backups by replicating its functionalities which are abstracted away from the applications. A distributed database is proposed for the use of cloud storage and redundancy is proposed for highly distributed unreliable systems. Local versions will run on the gateways providing local instances of data, smart migration and backup.

3.2.3. M2M communication and registration

Each gateway is equipped with its own set of communication mediums to transmit and receive messages from sensors and actuators. To account for differences in communication protocols and communication mediums the gateways have a driver for each medium that acts like a broker between the devices and the messaging system. These brokers are used to authenticate devices and add them to the locally available list for security and encryption. Furthermore, they interpret the received messages and assign the proper routing and header details to assure that they reach the required destinations. These drivers can have a more diverse range of tasks based on the requirements of the protocols and mediums such as packet forwarding, routing table creation and other WSN gateway tasks.

The registered device information is stored in the driver specific database and is used by the gateway controller to determine which application to deploy and for routing purposes. Furthermore, the device information saved in the database, that uniquely identifies the connected physical devices and their states, is used by the driver to monitor, authenticate and correctly route messages to their destinations.

Due to the wide range of protocols and transmission mediums, the messaging and routing system needs to be configured in such a way to allow different drivers to send and receive messages in a unified way. A slightly altered version of a JSON based markup language presented in [34] that has been used to link advanced IoT structures in [35], called Sensor Markup Language or SenML is proposed. This would require all connected devices to register, send and receive information based on this language. The device registration information needs to contain information about the device's type, its version, and the sensors that it is equipped with. Any other communication specifics that are not relevant to applications or monitoring of devices are abstracted.

The SenML based device message transmission is used for driver to driver, driver to app and app to driver communication only and is used as a common medium between protocols that can describe messages that need to be sent. The actual messages sent to the devices may vary depending on the control protocol. This would allow older devices to use their existing handshakes and means of message transmission to be connected to the system.

3.2.4. Application container

The client bundle in the container can be configured to read messages from a messaging service queue and to create events based on these messages. The applications create events, the broker reads the messages generated from these events and sends them on to the local messaging service as shown in Fig. 2. The headers of the messages are designed to allow applications to send messages to different locations, but also to act as a filter between the application container and the gateway resources only allowing applications to send messages to their pre-configured resources.

Communication between applications can be carried out in two distinct ways. The messaging system is more suited for communication between applications that are not closely linked to each other and can be interchanged. Communication through the internal services or other structures provided by the container is more suited for use within the same application set to create larger application from individual bundles following the Microservices architecture. The only constraint is that applications that communicate with each other through container specific structures need to be migrated together. Those which communicate through the messaging service can be kept in different locations and migrated separately.

In order to enable applications to respond to new devices being added to the system as well as to be able to listen to individual devices and have messages transmitted to these applications from the cloud, applications need to be able to reconfigure their application name and the name of the devices they are listening to. This is achieved through assigning a configuration file to each application that contains all the relevant information. The gateway controller adds information regarding the devices the application is configured to communicate with as well as the applications name, the regional communication channel, the cloud connections and other configuration parameters. When the configuration is updated, all applications are refreshed to start with the new set of data. Applications can then be deployed into multiple environments with multiple use cases as well as facilitating their testing and migration.

The construction of the application container allows for application migration within the local cluster and to the cloud. One of the main differences between the application container on the host or other local gateways and the cloud based/virtual gateways is the complexity of the messaging service. The gateways residing on the cloud only receive and send information from one source, having the modified brokers in the application container mimic the gateway sources of the messages based on the message headers. This difference allows the deployed applications to be location agnostic, receiving messages in the same format. Finally, using this method, the creation of virtual gateways is possible as well. These virtual containers have dummy applications that mimic the behavior of real devices by posting and consuming events on their behalf to allow for a more realistic testing environment as well as for scaling experiments.

3.2.5. Regional communications and clustering

Regional communication refers to gateways that are on the same network or can reach each other through local network scans or any other methods that send and receive application messages for clustering, high-availability or other inter-application communications. There are two ways to connect and access applications from the local network. The first one, with more constrained connection is realized through the federation of the messaging service, so gateways can connect to each other seamlessly. The second one is a more loosely coupled connection that would allow messages to be sent from different clients through the regional drivers that convert the messages to application messages inside the messaging service. The federation configuration of the messaging service offers better security, message latency and ease of use due to the fact that it extends the messaging service from one device to another by having the exchanges mirror on all nodes and having some of the queues unique to their specific gateways. Applications can be deployed on a single node and communicate with other devices and cloud tenants. The federation messaging approach also enables devices to configure clustering and high-availability as resources which may lead to better QoS parameters.

The more loosely coupled connection through the regional drivers would permit gateways to be of different types and configurations with even outside applications connecting to these endpoints. The configuration of these endpoints would be fulfilled by the cloud controller that creates a queue for each application that has regional communication set up in the configuration files and modifies the driver to make these available through external requests. These requests are treated as RPC calls and each request has a unique transaction id. This solution offers extra functionality and reduces costs by adding an alternative of accessing applications through the local network rather than through the cloud connections.

3.2.6. Cloud connection and management

Cloud connections enable the gateway to send and receive application data, sensor and actuator data, as well as to migrate applications through message passing and the deployment of applications. Each connection to the administration or tenant clouds is managed by a designated broker through the protocol preferred by the cloud. The first connection is to the main cloud, which is pre-configured in the gateway. The other connections can be started through commands received on the first one using the gateway controller.

When applications are migrated to the cloud, the respective connection is used to allow messages that would normally be transmitted to the local container to be transmitted to the cloud where they are routed to the cloud container. The brokers in the container transform them into messages with the headers and payload corresponding to those received on the physical gateways container. Applications can be migrated from the physical gateway to a virtual gateway in the cloud while retaining all inter-application communication and local messaging without reconfiguring or redeploying the applications.

In order to allow inter-application communication to occur a forwarder is required in the container that receives messages designated to the application and sends them to the cloud communication component as well as accepting responses and creating events as if the application was never migrated. This functionality can be extended to replicate local services on the container.

4. Architecture implementation

The existing architecture is implemented based on the general descriptions and technical requirements presented in Section 3. The implementation demonstrates the feasibility of the proposed generic gateway architecture as well as the use of the OSGI container as a gateway application container. The underlying messaging architecture is AMQP within the RabbitMQ server. The proposed communication mechanism with the cloud is MQTT which has received support from an increasing number of cloud providers. For the regional communication, either REST or STOMP based drivers are proposed while the clustering of gateways is supported through the federation functionality of the RabbitMQ messaging server. Each M2M communication protocol and device has its own functionalities, advantages and drawbacks. The device drivers' subsection shows the basic backbone to the drivers that were used. For the application container, the OSGI based Karaf is the most compliant with our generic architecture.

4.1. Device drivers

The approach to creating the drivers has been tested for 4 different communication mediums, 434 MHz, rf24, Bluetooth and Xbee.

These four mediums differ in their level of abstraction of the OSI layers as well as in their added functionalities. The first protocol only implements the physical level requiring the driver to configure the rest. The rf24 based protocol has the added functionality of discovery and being able to listen to specific channels, but what this lacks is the ability to listen and communicate on multiple channels effectively. The Bluetooth based RFCOM communication protocol allows for a wider range of features and sending messages through sockets to certain devices. Xbee is a similar protocol having multi-hopping and networking functionalities as well. The drivers for these protocols work in the same way for applications, with none of the differences being visible at the application level.

All applications to be deployed in the proposed architecture at least include a few common functionalities and these are: the registration of devices; the monitoring of devices; and the sending and receiving messages from devices based on their ids. The whole registration procedure is shown in Fig. 3 for the case where the device has been previously registered or when it is a newly registering device.

After the registration, devices only send a shortened version of the sensor data, only containing the sensor name, the value and the device id. The received message is parsed to make sure that it is consistent to the JSON format and key information like the device id is extracted and then the appropriate header information is created and the payload is sent to the messaging service. The structure of this message can be seen in the example shown in Table 2.

Information regarding the time when the message was received and the driver id is added. Furthermore, the device id is used to retrieve the device type and order from the registered device database and added to the headers to simplify routing and application development.

4.2. Application container

There are a number of candidates for the application container like Docker, that would allow applications of any type to be deployed and some for language specific application like Python and NodeJs, usually web-application deployment based on the Web Service Gateway Interface (WSGI). The container, which best fits our requirements as well as possessing extensive control of deployment and lifecycle management, was based on the Open Service Gateway Interface (OSGI) framework [36], which is designed for deploying modular java applications, dynamically on top of the Java VM. The Apache Karaf [37] implementation of the framework has a number of add-on libraries that are key components in the development of applications using the Microservice architecture and in enabling a wide range of applications to be deployed side by side.

Table 2
Message from driver.

Content name	Data/Property	
	Property name	Property value
Header	device	OWaDMY9V
	dev_type	ardUnoTemp
	dev_count	0
	comm	Gateway-RF24
	datetime	2016-05-09 12:02:36
Payload	[{"v": "26.00", "n": "temp"}, {"v": "34.00", "n": "hum"}, {"v": "8.95", "n": "dew"}]	

To allow applications to listen to specific events, the received messages are routed to the EventAdmin based on their headers and contained data. These routing rules can be seen in Table 3.

In order to allow for applications to respond to new devices being added to the system as well as to be able to listen to individual devices and have messages transmitted to these applications from the cloud, they need to be able to reconfigure their application name and the name of the devices they are listening to. This is achieved through the *ManagedService* class's *update()* function that allows applications to read the configuration file. In this case, each device will have its own file where the gateway controller adds information regarding the devices the application is configured to communicate with as well as the application's name, the regional communication channel and other configuration parameters. When the configuration is updated, all applications are refreshed to start with the new set of data. Applications can be deployed into multiple environments with multiple use cases to facilitate their testing and migration.

The applications can be managed through the gateway controller, while their status and performance are monitored through the bundles deployed on the container according to JSON based deployment files. For the migration of applications, the internal structures used for communication-like services allow for containers to migrate this service on the local region if configured properly, but for our implementation they are considered to be available only on the local deployment and applications linked through these services are required to be on the same gateway.

4.3. Regional and cloud drivers

The drivers used to connect to the cloud providers are designed to broker messages from the local AMQP messaging service to a

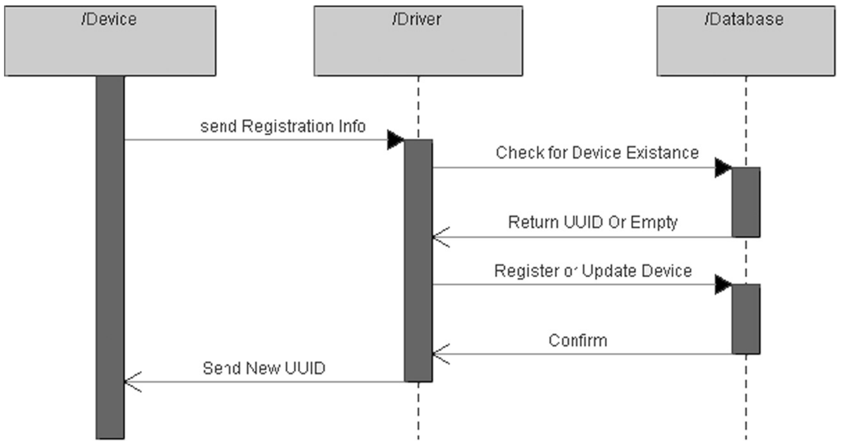

Fig. 3. Registration sequence diagram.

Table 3
OSGI message translation.

Sender	Key property	Receiver	Resulting topic
device	dev_type	app	/device/receive/[dev_type]
app	*	device	/device/send/
cloud	app_name	app	/cloud/receive/[app_name]
app	*	Cloud	/cloud/send/[app_name]
resource	resource_type	app	/resource/[res_type]/receive
app	resource_type	resource	/resource/res_type]/send
region	app	app	/region/receive/[app_name]
app	*	region	/region/send
app	app_name	*	/apps/[app_name]/send
*	app_name	app	/apps/[app_name]/receive

Message Queue Telemetry Transport (MQTT) server hosted on the cloud. MQTT was chosen as the connection protocol to the cloud due to the wide range support from major cloud providers like AWS and the added functionalities these propose. This lightweight messaging format was designed for high-latency or unreliable networks so it offers the best solution for asynchronous messaging between cloud and gateway. The cloud communication drivers are designed to allow for single direction connections and can have multiple providers connected and routed through their instances.

Connecting to the cloud can be done through the Secure Socket Layer (SSL) or through simple username and password authentication, depending on the security requirements and the provider's options. In order to send and receive information from the cloud messaging service, the proposed broker translates the byte-array messages into headers and payloads as well as sending them to the required exchange. The first connection is to the main cloud, which is done before the gateway starts. The other connections can be started through commands received on the first one using the gateway controller.

Messages on the local cloud queue need to be parsed into a byte array and sent to the cloud. The solution for this parsing problem is creating JSON strings from the received AMQP messages where each header and the payload are made into a JSON object. The payload is either parsed as one object, or as sub-components formatted in JSON as seen in Table 2.

Drivers used for regional communication use REST APIs to receive and send messages from applications. Each application has the option of configuring one or more regional connections that can be used by outside applications. These are configured by creating a queue for each and routing the queues based on URL location on the REST APIs which get configured by the gateway controller. This configuration would allow applications to have their own access keys and authentication option on the region. The other proposed drivers would rely on STOMP messages being routed to the messaging service based on correctly formatted headers. Messages with the appropriate configuration would be routed and those without the right data would be lost.

5. Deployed architectures and scenarios

The aim of this section is to show two use case scenarios that highlight the novel aspects and features of the platform that are not present in other systems. Each deployment scenario is designed to demonstrate a set of functionalities. The first scenario shows a distributed control and metering application that exhibits the advantages of migration and clustering within a home and office environment. The second scenario looks at an industrial deployment use case where the focus is more on the development, testing and deployment of industrial control environments through the gateway platform, which showcases the use for virtual gateways and virtual devices connected to these gateways. These functionalities are missing or need extensive programming to be able to be implemented in other systems, which can cause a longer reconfiguration and redeployment time for factories that have a changing development environment, while in the home and office setting, they provide a highly compatible platform that can communicate with a wide range of devices and deploy applications anywhere needed in the region. These use cases are discussed and analyzed in depth in the following subsections where the deployment details, simulation scenarios and comparisons are shown.

5.1. Distributed control and metering application

The adopted use case scenarios show a simple home and office monitoring and a control environment where the advantages that the advancements of our system brings over other platforms are highlighted. The deployment scenario looks at cases where gateways are deployed in multiple rooms or environments having their own set of devices connected to them. Our platform allows for the deployment of applications to a variety of locations while making use of these resources without the need to reprogram the applications. Furthermore, these applications can communicate with a wide variety of devices through the same format regardless of the devices communication protocol. This allows for a better use of resources on the gateways as well as the deployment of complex applications on a singular gateway that make use of resources throughout the system. These functionalities would be very difficult or impossible on the existing platforms.

The setup has the configuration of a smart home thermostat. It consists of a humidity and temperature sensor, a presence sensor and an actuator device that turns the heating on and off. The application reads data from the devices and controls the actuator based on a control algorithm while saving all relevant temperature readings to the database and all important events to the log. The applications are able to receive user commands locally or from the cloud. The setup also allows for the applications to be migrated among gateways and to the cloud based on the configuration needed. A deployment of the system can be seen in Fig. 4 where the first application is deployed on the cloud while the second one is deployed on the RF24 capable gateway.

The first application has the task of collecting the sensor data, saving it to the database and the log files and sending periodic reports to the second application. The second one is tasked with reading reports from the first application and comparing those to its control algorithm and sending control signals to the actuator, while saving information to the log files and sending reports to the second cloud connection. These two applications can be deployed anywhere on the two gateways or the cloud, being able to control and read the devices while performing the logging and storage tasks.

The three devices have their unique sensors and tasks, and are designed in a way that they can be analogous to low power sensors. The two sensing devices are Atmega attiny85 boards equipped with nRF24L01 transceivers, one of them with a temperature sensor and the other with a light based presence sensor. The actuator is an atmega128rfa1 434 MHz communication enabled microcontroller that signals a relay that controls the heating agent. These devices communicate with the gateways through the dedicated drivers. An nRF24L01 transceiver is connected to the GPIO pins of the Raspberry Pi. The heating actuator is connected to the first Raspberry Pi, while the temperature and presence sensors are connected to the second one.

The implementation shows the clustering and migration functionality of our platform as well as scenarios where applications may need to communicate with devices connected to other gateways on the region and how this is deployed. Finally, we

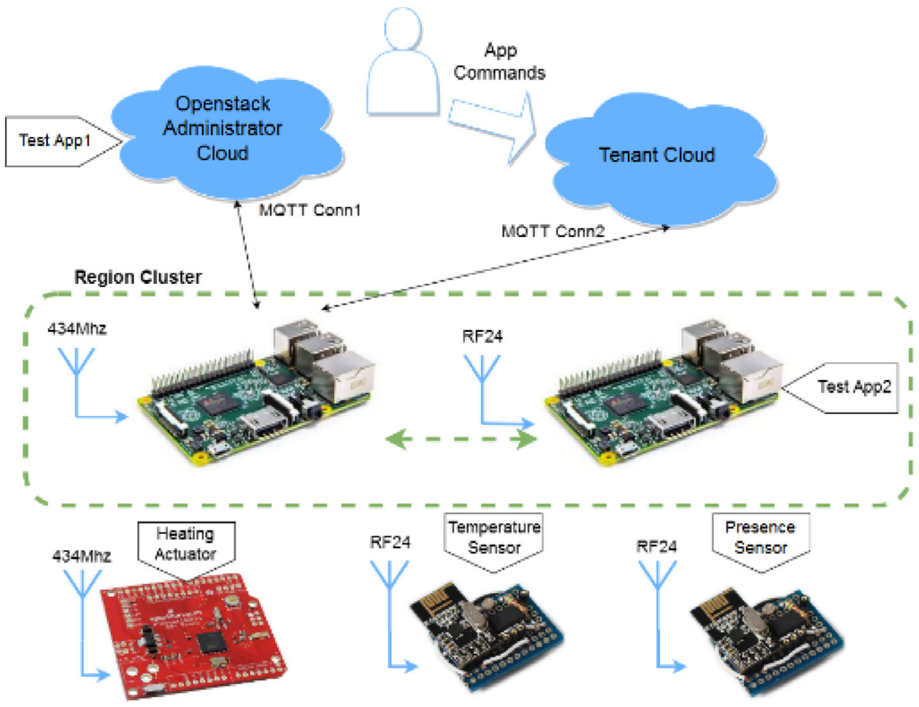

Fig. 4. Control and metering application.

demonstrate that the presented implementation shows functionalities and scenarios which surpass the capabilities of other systems, due to migration and message passing between gateways as well as presenting a proposed use case for the system.

5.2. Milling machine control and simulation

The implemented use case scenario shows a more complicated industrial monitoring and control environment where the functionalities and possibilities of such a system are explored. The proposed platform is used as a back-end to a milling machine task deployment and monitoring system. This system looks at the use of virtual gateways to test the deployment of applications as well as the scaling of the system. Our platform allows for the migration of applications into virtual containers where devices can be replaced with dummy applications that respond in the same way as the physical counterparts. The virtual gateways allow applications to be extensively tested before deployment as well as for virtual gateways to be deployed to mimic the scaling of the system to test for faults. These functionalities are missing or harder to implement in other platforms due to the tight coupling between applications and their gateways.

This implementation consists of a cloud environment that receives tasks from end users and sends them to the required gateways where these tasks are processed and sent to the machines while metering information is collected. Furthermore, a simulation environment is designed to allow test applications to be deployed on the devices as well as simulating the runtime of these devices. In Fig. 5 the overview of this system can be seen where we consider 3 connected Computer Numeric Controlled (CNC) milling machines and 3 virtual devices each running in its own Karaf container with the respective control applications.

The cloud orchestrator has two distinct tasks. The first is to receive milling jobs in the form of G-code from the users, save them to the task list and schedule these jobs to the appropriate devices. The second is to orchestrate and monitor devices. This is done by sending them the required milling jobs and receiving metering and status information. The scheduling of devices is done by getting the oldest job, finding the free devices that have the correct constraints and sending these jobs to the fastest free devices, after which they are marked as assigned. The cloud controller component receives messages from the gateways and performs updates based on these or saves the metering data to the database while receiving job start commands from the scheduler and sending these to the appropriate gateways and devices. The communication with the devices is done through the messaging service which routes the messages to the appropriate gateway queue.

There are two types of gateways used for this scenario. The first is the physical one that works as presented and has one Arduino controlled milling machine attached to it running an open-source milling machine control algorithm together with the rf24 communication protocols. This gateway also has two other Arduino boards connected with the same application but lacking the control drivers. The virtual gateway has three applications deployed to the container that receive the messages intended for devices and return answers with a delay and message loss similar that of the actual devices.

Orchestrating the devices on a gateway level is done by the milling machine control application which reads the device list from the configuration files, configures the appropriate event listeners and creates the required objects. When a G-code file is sent to the application, it creates a new job for the specified device and sends back an acknowledgement of receiving the data and some metadata information. After the start command is received, the gateway continues sending command lines to the devices until it is stopped, an error occurs with the G-code, or the file is finished. After this, a finish message is sent to the cloud. The application also handles retransmission of data in case no response from the device is received. The same application runs on the physical machine and on the virtual one as well.

In order to create dummy devices we need to monitor the existing devices and gather information on how they transmit and receive data and how much time it takes to process commands. This is done through a monitoring application that looks at when

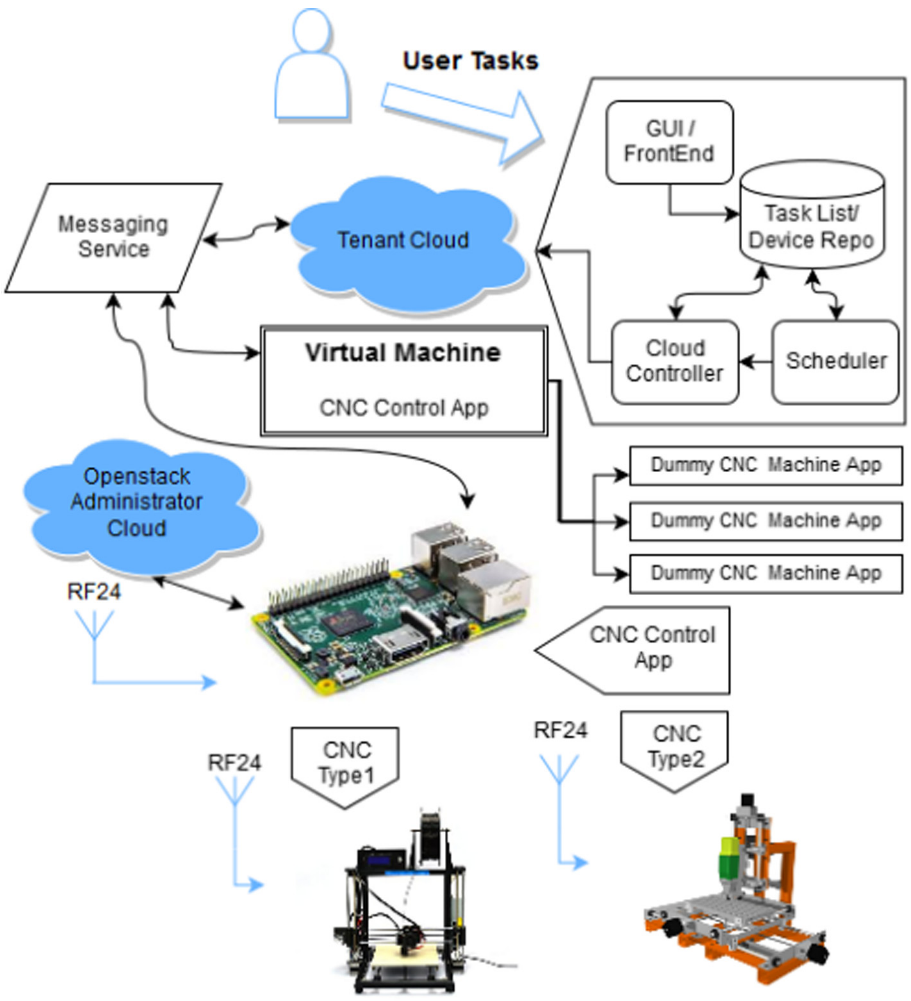

Fig. 5. Overview of orchestration system.

a message is sent, how much time it takes to receive a response and how many retransmits needed to be sent in order to receive a response. The summary of this information can be seen in Table 4.

The comparison of the two devices' responses can be seen in Fig. 6.

We can see that there is a big gap between the minimum and maximum times, which is due to how the devices work. They receive a message, perform the action and send an acknowledgement afterwards. Because of this behavior, some actions like setting the parameters from inches to mm respond in 42 ms, while longer tasks like resetting the machine will last for a longer time. Other than this, the response times are similar and the retransmits are close as well.

A test of 20, 25 line tasks was deployed to these applications each of them having 1 device running. The results of these can be seen in Fig. 7.

Table 4
Physical and simulated device responses.

Attribute	Physical device[a]	Simulated device[a]
Mean response time (ms)	268.58	306.45
Min. response time (ms)	42.82	47.54
Max. response time (ms)	949.11	1018.27
Retransmit number (cnt)	4	3

[a] Data collected for a 100 line G-code file.

The graph shows that even though from the experimental data, seen in Table 4, the virtual device is slower than the physical device, when 20 tasks were deployed the virtual device finished faster by 19 s for the presented set of tasks. The physical device finishes in 233 s and the virtual one in 217 s. The difference in the behavior of these gateways lies in the latencies caused by the messaging service and network on which the two devices reside. The Raspberry Pi is configured to add latency to its connection to the cloud to mimic a real-life environment while the virtual machine is inside the cloud where the latency between hosts can be lower than 1 ms. Furthermore the messages from the virtual device are directly transferred to the cloud messaging queue while the ones on the physical device are first routed locally and then sent up to the cloud through Python based drivers that can cause a bottleneck. These differences can be solved by altering the communication apps for the Karaf container to mimic the latency and other characteristics of the ones on the physical device.

The evaluation of the implemented platform shows us that the testing of an application before deployment in a virtual gateway is possible and the characteristics of devices can be modeled. Furthermore, these virtual gateways can be attached to real systems to simulate increased load which would allow for a more complete testing. Finally, the implemented application shows that higher QoS requirements can be implemented through the retransmission of messages and the monitoring of the system for faults. These results show that the system can be used in an industrial environment in both testing and control. Finally, this system shows the

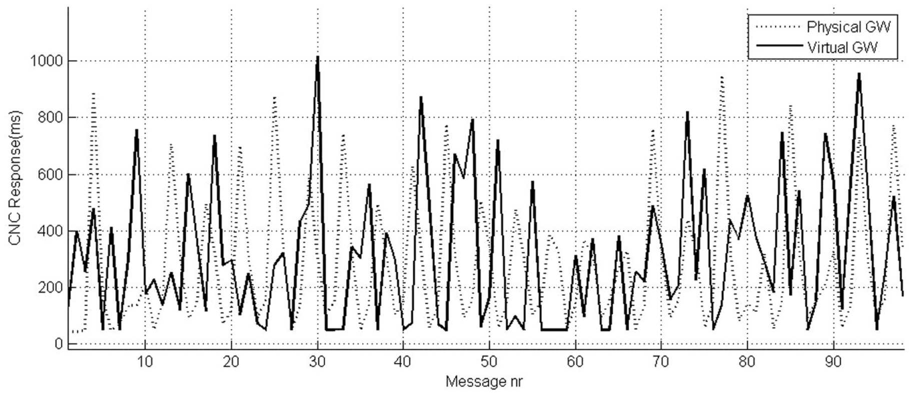

Fig. 6. Device response times comparison overview of orchestration system.

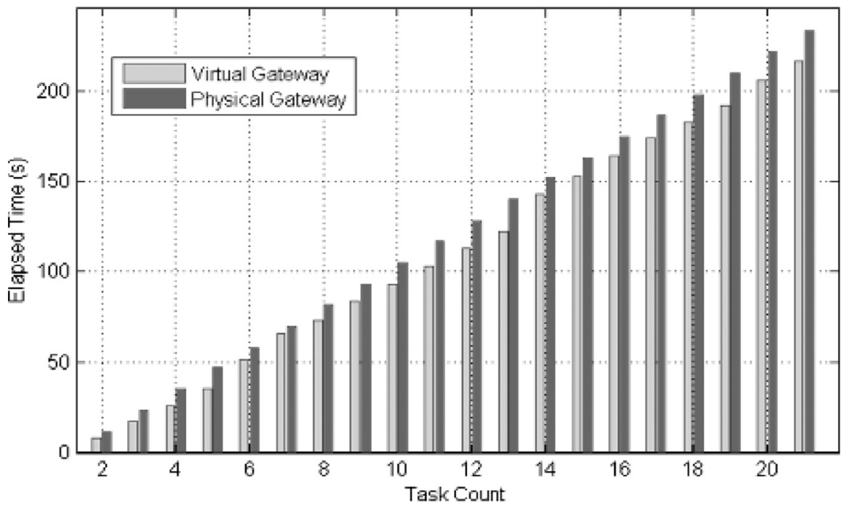

Fig. 7. Virtual and physical task processing.

possibility of simplifying the orchestration tasks on the cloud by deploying these to the gateways, delegating monitoring and control.

6. Performance evaluation

The evaluation of the presented platform was done from a QoS point of view to analyze whether the functional improvements that were made to existing systems would result in a better performance parameters. The parameters that were taken into account were variance and response time of device messages. The advantages of the proposed migration of applications to the cloud are analyzed also. The evaluation shows when such a migration is needed and what QoS parameters improve or deteriorate as well as what parameters are not affected. These tests do not take into account network usage, the cost of connection to the cloud and the power usage differences in the three different scenarios.

The cloud platform used for testing is an Openstack Kilo deployment running two VM's with 4 GB of Ram and 2 processors with a ping latency between them of 0.04 ms. The Gateway is a Raspberry Pi 3 which is required to run the server virtualization of Karaf that has 1 GB of Ram and 1.2 GHz processor that is connected to the Openstack network. A real-world situation is simulated by adding latency to the network by using the Netem network emulator to add a delay range of 15–30 ms with a normal distribution. The benchmarks used for testing REST, COAP or MQTT servers were not appropriate for this test due to their focus on throughput and their performance under high number of connection scenarios. The testing method we used is designed to focus on the response time of commands while adding load to the applications based on the IBM benchmark suggested in [38].

The testing environment has three scenarios tested: (a) the direct connection, labeled Dev-Cloud looks at a smart device directly connected to the cloud, sending and receiving messages through MQTT in a Karaf environment: (b) a scenario where the device is connected through an nRF24L01 transceiver to the gateway and the application is on the gateway; and (c) the third scenario has the device connected to the gateway which forwards messages to the cloud where the application is deployed in the same way as in the first scenario. These scenarios and their notations are described in Table 5. The Dev-Cloud scenario serves as a baseline providing the ideal scenario for a Smart Object approach to metering and control).

The benchmark application has four settings. The first one contains no extra processing load, which means that it receives the data, adds the numbers up and sends it back, this representing a very simple application. The following settings are based on the benchmark in [38] that adds floating point operations to mimic processor load. The setting had the floating point operations performed 10^6, 10^7 and 10^8 times with the results visible in Fig. 8.

The results of the experiments show that the scenario with no extra load, the Dev-Gateway deployment has the lowest response time out of the three, with a 20.14% reduction compared to the Dev-Cloud case and a 38.33% reduction compared to the Dev-

Table 5
Testing connection types and attributes.

Connection name	Application location	Message route[a]
Dev-Cloud	Cloud	D-C-D
Dev-Gateway	Gateway	D-Gw-D
Dev-Gw-Cloud	Cloud	D-Gw-C-Gw-D

[a] D - Device; C - Cloud; Gw – Gateway.

Table 6
Processing time in µs.

Iteration Nr.	Gateway	Cloud
10^6	756	188
10^7	8681	1346
10^8	76548	9806

Gw-Cloud one. This advantage can be seen on the second test with 10^6 iterations of the load process, but with a decrease of the advantage to 15.38% and 35.65% respectively. This advantage is reduced further in the 10^7 tests where the Dev-Gateway case is slower than the Dev-Cloud one by 6.04%. The Dev-Gateway case has the worst performance in the 10^8 test where the deployment is slower than the Dev-Cloud case by 159.79% and slower than the Dev-Gw-Cloud option by 107.02%. The average processing times on the container are shown in Table 6 and we can see the evolution of the response times in Fig. 9.

The figures are reflected in Fig. 9 we can see that the gateway is a better choice for the low load situations, but when the processing requirements increase, the gateway configuration poses a real disadvantage to the system, validating the need for migration.

As mentioned in the beginning there are certain aspects of QoS that we do not take into consideration for these tests. It is worth noting that from the point of view of network load, number of messages and robustness, the local messaging is much more stable, especially if we compare with other use cases such as high-latency and high loss internet connection. Furthermore, deploying applications to the gateway would reduce the communication with the cloud significantly, even if we consider the deployment task.

The tests were carried out to compare the two scenarios, where the application is deployed on the gateway and when it is on the cloud, to a baseline which is a smart device with direct connection to the cloud. This was done in order to measure response time and its variance between these two systems as well as compare them to a best case scenario for smart devices. The tests show that with small processing loads, less than or equal to 10^7 flops, the deployment of the application on the gateway makes for a better performance for both response time and variance. When the gateway is subjected to loads greater than 10^7 flops both the response time and its variance deteriorate to an extent where they fall below the baseline and make migrating the applications a better choice from the two QoS parameters' perspectives. We can see that the variance has a sharp increase at this mark which might be accounted for by the processing time differences overloading the gateway can cause. Based on these tests we can conclude that for certain applications there is a real need for migration and to allow load balancing throughout the cluster.

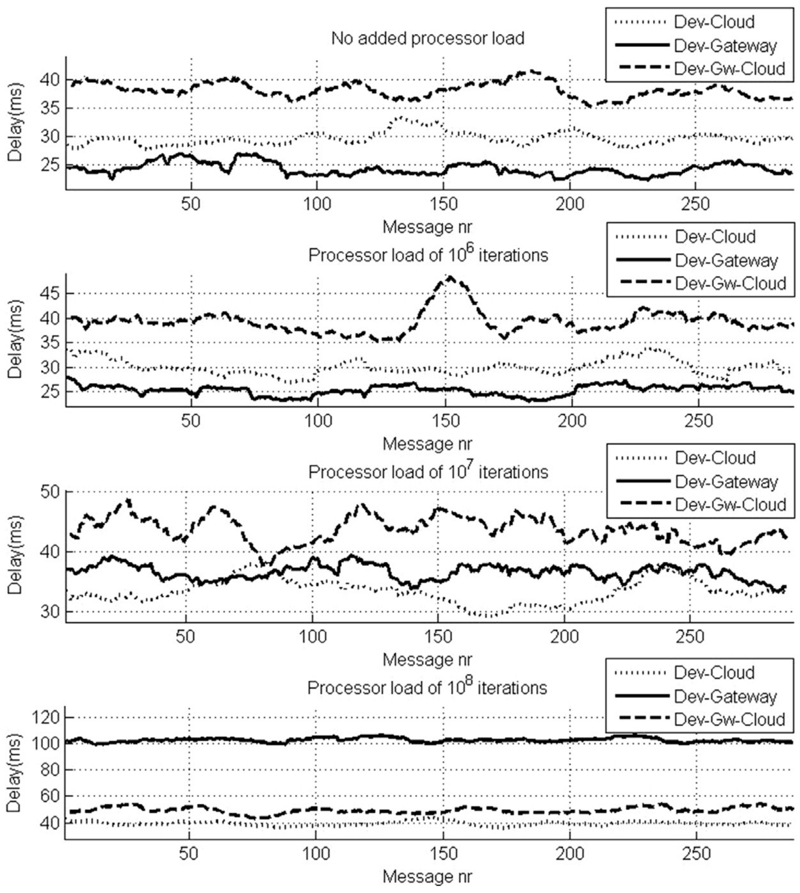

Fig. 8. Processor load based response time.

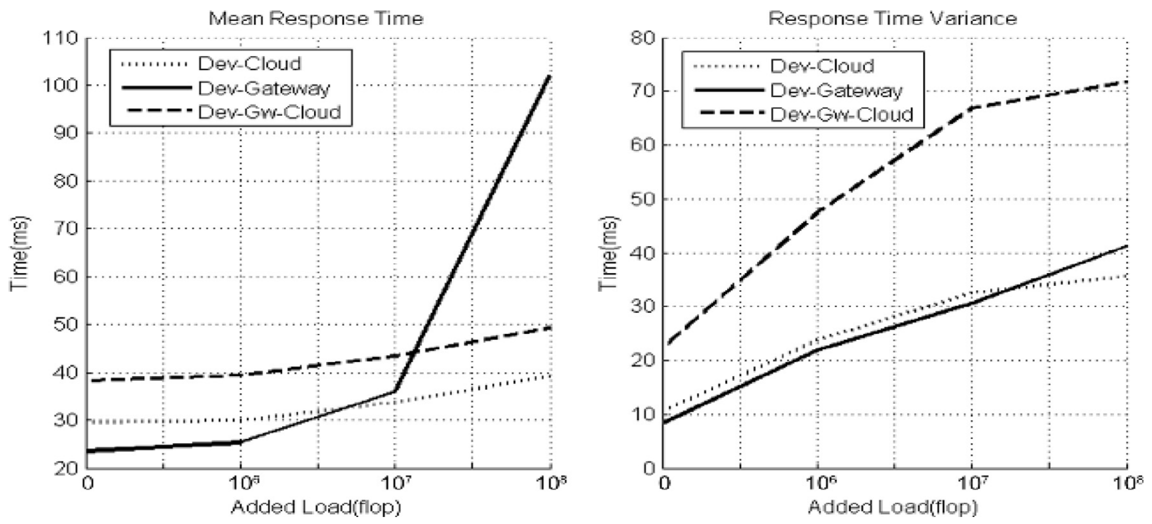

Fig. 9. Response times and variance.

7. Conclusion and future directions

To be able to integrate computing in our everyday environment and have a ubiquitous computing and sensing environment through the Internet of Things we need to have a homogeneous network of devices that can be connected and controlled from a wide range of platforms. For increased QoS parameters like security, latency, network load and data availability, the use of gateways has been suggested, and through the paradigms of Fog Computing the resources of these gateways would be better utilized to process, store data and route functionalities locally.

This paper proposes an IoT platform and gateway that allow applications to be deployed closer to the network edge and migrated to the cloud based on the users' requirements. Using the OSGI gateway for application deployment allows lifecycle management of applications as well as the deployment of a set of applications as they can work together in a Microservice environment. Furthermore this solution allows parts of the applications to be migrated to the cloud where the more processor intensive tasks might be performed. In addition, an evaluation of the latency and the processing power of such a system was carried out and presented.

The system was tested for different use case scenarios with an extensive latency/processing test done on the raspberry pi and cloud environment. The home and office use case explores the modular deployment of applications to create a smart home environment. The milling machine control scenario looks at the orchestration and simulation of industrial machines through the platform, ensuring messages transmission to the device while maintaining low latency.

Compared to similar research in the field, this platform, through dynamic abstraction, allows for a protocol agnostic application environment, as well as a modular deployment of applications to the gateway. The platform also provides a solution for the increased horizontal integration of devices by allowing multiple tenant connections to be configured from the gateway that may use resources available from different providers. It also provides for speedy creation of test environments and the option of migrating between cloud and gateways on the region depending on processing needs. Furthermore, the gateway makes steps towards a better horizontal integration by allowing the connection through different drivers to local and cloud resources while allowing different application environments and device connections. In an industrial environment this would allow for faster time to market, a more dynamic production environment, faster software upgrades and easier testing.

Future work on this topic will consist of developing a smart laboratory environment with more diverse and complex devices and control scenarios to fully test the system. Work on the architecture will include a metering system that follows the resource use of applications both in the cloud and on the gateway, as well as a constraint and optimization based deployment of applications from a repository. Furthermore, the optimization algorithms will be assessed to see whether the use of these can improve the overall QoS of the system. One of the drawbacks of the system that it is not designed to process data streams like sound and video which are increasingly in use. Further research will be done to investigate deploying components for image and sound processing within the proposed architecture. Finally, we will look into the use of hosting multiple containers with different programming languages on single gateways as well as on the cluster. We will also investigate data persistence throughout migration. Another direction of this work will focus on service-oriented fog and cloud computing [39,35] to facilitate the integration and interoperability.

References

[1] Joshua Cooper, Anne James, Challenges for database management in the internet of things, IETE Tech. Rev. 26 (5) (2009) 320–329.
[2] Lasi Heiner, Fettke Peter, Kemper Hans-georg, Feld Thomas, Hoffmann Michael, Industry 4.0, Bus. Inform. Syst. Eng. 6 (4) (2014) 239–242.
[3] Alastair Watson, Digital buildings – challenges and opportunities, Adv. Eng. Inform. 25 (4) (2011) 573–581, ISSN 1474-0346.
[4] Rob van der Meulen, 6.4 Billion Connected "Things will be in use in 2016", Gartner http://www.gartner.com/newsroom/id/3165317, STAMFORD, Conn., November 10, 2015.
[5] Zhuming Bi, Li Da Xu, Chengen Wang, Internet of things for enterprise systems of modern manufacturing, IEEE Trans. Ind. Inform. 10 (2) (2014) 1537–1546.
[6] Fei Tao et al., IoT-based intelligent perception and access of manufacturing resource toward cloud manufacturing, IEEE Trans. Ind. Inform. 10 (2) (2014) 1547–1557.
[7] Fei Tao et al., CCIoT-CMfg: cloud computing and internet of things-based cloud manufacturing service system, IEEE Trans. Ind. Inform. 10 (2) (2014) 1435–1442.
[8] Ben Mitchinson, Tak-Shing Chan, Jon Chambers, Martin Pearson, Mark Humphries, Charles Fox, Kevin Gurney, Tony J. Prescott, BRAHMS: novel middleware for integrated systems computation, Adv. Eng. Inform. 24 (1) (2010) 49–61, ISSN 1474-0346.
[9] Giancarlo Fortino, Antonio Guerrieri, Wilma Russo, Claudio Savaglio, Middlewares for smart objects and smart environments: overview and comparison. internet of things based on smart objects, Technol., Middlew. Appl. (2014) 1–27.
[10] Pablo Punal Pereira et al., Enabling cloud connectivity for mobile internet of things applications, in: 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), IEEE, 2013.

[11] Geoffrey C. Fox, Supun Kamburugamuve, Ryan D. Hartman, Architecture and measured characteristics of a cloud based internet of things., in: 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), IEEE, 2012.
[12] Chayan Sarkar et al., DIAT: a scalable distributed architecture for IoT, Internet Things J., IEEE 2 (3) (2015) 230–239.
[13] Kuo-Ming Chao et al., Cloud E-learning for mechatronics: CLEM, Future Gener. Comput. Syst. 48 (2015) 46–59.
[14] Christian Inzinger et al., MADCAT: a methodology for architecture and deployment of cloud application topologies, in: 2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE), IEEE, 2014.
[15] M. Kovatsch, Y.N. Hassan, S. Mayer, Practical semantics for the Internet of Things: physical states, device mashups, and open questions, in: 2015 5th International Conference on the Internet of Things (IOT), Seoul, 2015, pp. 54–61.
[16] Farzad Khodadadi, Rodrigo N. Calheiros, Rajkumar Buyya, A data-centric framework for development and deployment of Internet of Things applications in clouds, in: 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), IEEE, 2015.
[17] Peter Ruckebusch et al., GITAR: generic extension for Internet-of-Things Architectures enabling dynamic updates of network and application modules, Ad Hoc Netw. 36 (2016) 127–151.
[18] Giancarlo Fortino, Antonio Guerrieri, Wilma Russo, Claudio Savaglio, Integration of agent-based and cloud computing for the smart objects-oriented iot, in: Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, 2014.
[19] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (SIGCOMM) (MCC'12), ACM, New York, NY, USA, 2012, pp. 13–16.
[20] Bukhary Ikhwan Ismail et al., Evaluation of docker as edge computing platform, in: 2015 IEEE Conference on Open Systems (ICOS), IEEE, 2015.
[21] Michael Vögler et al., A scalable framework for provisioning large-scale iot deployments, ACM Trans. Internet Technol. (TOIT) 16 (2) (2016) 11.
[22] Michael Vogler et al., DIANE-dynamic IoT application deployment, in: 2015 IEEE International Conference on Mobile Services (MS), IEEE, 2015.
[23] Afkham Azeez et al., Multi-tenant SOA middleware for cloud computing, in: 2010 IEEE 3rd International Conference on Cloud Computing (Cloud), IEEE, 2010.
[24] Sangwon Seo et al., HePA: hexagonal platform architecture for smart home things, in: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2015.
[25] Arne Koschel et al., Asynchronous messaging for osgi, J. Comput. Inform. Technol. 20 (3) (2012) 151–157.
[26] Alessandro Sivieri, Luca Mottola, Gianpaolo Cugola, Building Internet of Things software with ELIoT, Comput. Commun. (2016).
[27] Peishun Ho, Amy J.C. Trappey, Charles V. Trappey, Data interchange services: use of XML hub approach for the aerospace supply chain, Int. J. Technol. Manage. 28 (2) (2004) 227–242.
[28] S. Veera Ragavan, Kusnanto Ibrahim Kusumah, Ganapathy Velappa, Service oriented framework for industrial automation systems, Proc. Eng. 41 (2012) 716–723.
[29] Ala Al-Fuqaha et al., Toward better horizontal integration among IoT services, IEEE Commun. Mag. 53 (9) (2015) 72–79.
[30] BUTLER Project. <http://www.iot-butler.eu>.
[31] eTrice – Real-Time Modeling Tools. <http://www.eclipse.org/etrice/>.
[32] Stefan Kimak, Jeremy Ellman, Performance Testing and Comparison of Client Side Databases versus Server Side, Northumbria University, 2013.
[33] Aazam Mohammad, Eui-Nam Huh, Fog computing and smart gateway based communication for cloud of things, in: 2014 International Conference on Future Internet of Things and Cloud (FiCloud), IEEE, 2014.
[34] Cullen Jennings, Jari Arkko, Zach Shelby, Media Types for Sensor Markup Language (SENML), IEFT, 2012. <https://tools.ietf.org/id/draft-jennings-senml-08.txt>.
[35] Harald Lampesberger, Technologies for Web and cloud service interaction: a survey, Serv. Orient. Comput. Appl. 10 (2) (2016) 71–110.
[36] OSGi Alliance et al., Osgi Service Platform, Release 3, IOS Press, Inc, 2003.
[37] Achim Nierbeck et al., Apache Karaf Cookbook, Packt Publishing Ltd, 2014.
[38] Brent Boyer, Robust Java Benchmarking, Part 1: Issues, IBM, 2008 [Online].
[39] Chengyuan Yu, Linpeng Huang, A Web service QoS prediction approach based on time- and location-aware collaborative filtering, Serv. Orient. Comput. Appl. 10 (2) (2016) 135–149.