A Security Policy Enforcement Framework for Controlling IoT Tenant Applications in the Edge

Phu H. Nguyen
Software and Service Innovation, SINTEF Digital
Oslo, Norway
phu.nguyen@sintef.no

Phu H. Phung
Department of Computer Science, University of Dayton
Dayton, OH, U.S.A.
phu@udayton.edu

Hong-Linh Truong
Faculty of Informatics
TU Wien, Vienna, Austria
hong-linh.truong@tuwien.ac.at

ABSTRACT

In the context of edge computing, IoT-as-a-Service (IoTaaS) with IoT data hubs and execution services allow IoT tenant applications (apps) to be executed next to IoT devices, enabling edge analytics and controls. However, this brings up new security challenges on controlling tenant apps in IoTaaS, whilst the great potential of IoTaaS can only be realized by flexible security mechanisms to govern such applications. In this paper, we propose a Model-Driven Security policy enforcement framework, named MDSIoT, for IoT tenant apps deployed in edge servers. This framework allows execution policies specified at the model level and then transformed into the code that can be deployed for policy enforcement at runtime. Moreover, our approach supports for the interoperability of IoT tenant apps when deployed in the edge to access IoTaaS services. The interoperability is enabled by an intermediate proxy layer (gatekeeper) that abstracts underlying communication protocols to the different IoTaaS services from IoT tenant apps. Therefore, our approach supports different IoT tenant apps to be deployed and controlled automatically, independently from their technologies, e.g. programming languages. We have developed a proof-of-concept of the proposed gatekeepers based on ThingML, derived from execution policies. Thanks to the ThingML tool, we can generate platform-specific code of gatekeepers that can be deployed in the edge for controlling IoT tenant apps based on the execution policies.

ACM Classification Keywords

D.4.6 OPERATING SYSTEMS: Security and Protection; D.2.2 SOFTWARE ENGINEERING: Design Tools and Techniques; D.2.11 : Software Architectures; D.2.12 : Interoperability

Author Keywords

IoT, Model-Driven Security, Access Control, Edge computing, ThingML, services computing.

IOT '18, October 15–18, 2018, Santa Barbara, CA, USA

ISBN 978-1-4503-6564-2/18/10…$15.00

DOI: https://doi.org/10.1145/3277593.3277602

INTRODUCTION

With the rapid development of IoT infrastructures and platforms, computing power and computing model have been moving towards the Edge [21]. Edge computing concepts and IoT infrastructures enable IoT as-a-service (IoTaaS), in which third-party IoT applications (called IoT tenant apps for short), carrying IoT data analytics and controls, can be deployed in IoTaaS platforms. This model brings up new security challenges to govern how IoT tenant apps use IoTaaS platforms because the traditional security and access control mechanisms can only provide secure communication or enforce coarse-grained policies. The great potential of IoTaaS platforms can only be realized by dynamic, flexible, yet manageable and easy-to-use security mechanisms to control IoT tenant apps. For instance, in IoTaaS scenarios, authentication and authorization are not as straight-forward as responding yes or no to access requests of IoT tenant apps according to a fixed policy. In practice, these scenarios require various dynamic aspects, such as, time, physical resources, and payment models, and other security aspects that must be considered at runtime to enforce IoT security policies. In other words, IoT tenant apps on IoTaaS platforms requires dynamic context-based policies.

Moreover, deploying and controlling tenant apps must consider their technologies to ensure that these apps are not only well supported to run, but also well controlled in terms of execution and security. If tenant apps are using different technologies from the *base IoT services of IoTaaS platforms* (IoTaaS services), we must ensure the interoperability among tenant apps and the IoTaaS services in the edge.

Problems: 1) IoT tenant apps must be controllable in IoT Edge servers whilst allowing them to use IoTaaS platforms. 2) Execution policies for IoT services and devices must be dynamic context-based policies, local and platform-specific. IoTaaS providers still need to have a centralized control and generic platform to specify, and then enforce execution policies in the edge servers. A centric and systematic IoT execution policies management and enforcement solution is vital.

Goals and the proposed solution: Because edge operations are very much context-sensitive, the design of security in IoTaaS should include support for dynamic context. Moreover, dynamic context normally also means local-specific, platform-specific context of IoT scenarios. On the other hand, from the engineering point of view, IoTaaS providers would go for a generic framework that allows having a generic, centralized so-

lution of specifying and developing generic execution policies, which will be enforced at runtime with dynamic context. In this work, we propose a domain-specific language (DSL) for defining generic execution policy for IoT. Our DSL leverages Access Control and Usage Control concepts [2, 17], with a focus on dynamic context for IoT scenario. The DSL is part of our model-based security engineering approach for IoT. Model-based engineering is gaining more favor as a way to engineer complex systems, such as IoT and Cyber-Physical Systems, especially including crosscutting concerns like security [14]. Along this line, our model-based security engineering approach consists of a model-driven framework to automatically enforce the policies defined with our DSL at a high level of abstraction into local-specific, platform-specific context of IoT scenarios. Moreover, our security policy enforcement framework implements some of the key Cloud Design Patterns such as the Gatekeeper and Throttling patterns [9] in combination with usage control [17].

Contributions: Our main contribution is the generic *Model-Driven Security (MDS) enforcement framework for IoT* (MDSIoT) in the edge. The MDSIoT framework provides a centralized execution policy management for IoTaaS providers. Empowered by an MDS approach, the framework allows generating security enforcement code, called gatekeepers, for different kinds of IoT tenant apps, and deploying a tenant app with its corresponding gatekeeper in the edge server. Moreover, MDSIoT supports for the interoperability of tenant apps when deployed in edge servers. In certain cases, tenant apps do not need to care about the underlying communication protocols to the IoTaaS services provided by the IoTaaS provider. Generated gatekeepers will play as an intermediate layer between the tenant apps and the IoTaaS services for enforcing execution policy in the edge. We demonstrate our approach with an initial prototype using ThingML [8].

In the remainder of this paper, first, we provide motivating examples and research challenges of our work. Then, we present our approach to tackling the challenges. After that, we discuss our current prototype, followed by related work. Finally, we conclude the paper and present the future work.

BACKGROUND AND MOTIVATION

A motivating scenario

Let us consider the case of IoTaaS having IoT data streams and edge servers for tenants to perform data analytics in the edge (Figure 1). Our case, derived from scenarios in the INTER-IoT project[1], is in the seaport management and logistics in which various data streams from sensors deployed in terminals, gateways, lanes, etc., in the seaport are aggregated into edge servers. Each edge server has a mini IoT data hub (running with MQTT/Mosquitto) for data streams. The sensors monitoring data about equipment, cranes, trucks, lights, electricity grids and the environment in the seaport are provided for third parties (e.g., truck companies, vessel management companies). The third parties will deploy their apps in the edge server to perform various tasks. The main stakeholders are as follows. The IoTaaS providers making data and edge computing nodes are the telcos/IT companies doing the seaport monitoring. Some examples of the tenants using IoTaaS are the seaport authority, truck companies, and vessel management companies. The tenants want to run their apps, called `TenantApps`, in the edge server, and access the `IoTaaS services` provided by the provider. The `IoTaaS services`, in this case, are `IoTDataHub` for providing data used by the `TenantApp` and `ExecutionEngineService` for running the `TenantApp`.

One of the first problems is that a `TenantApp` can only be allowed to access certain types of data because there are many tenants using the same `IoTDataHub`. For example, the tenant maintaining lights is not allowed to access data about gate controls, which are performed by another tenant. Furthermore, each tenant can also use the `ExecutionEngineService` for executing their work-flows or dockerized analytic programs with certain constraints based on their contracts. Note that one tenant might run the same `TenantApp` on multiple edge servers, for example, when the port management has to control lights based on truck accesses.

From the IoTaaS infrastructure provider's viewpoint, the provider does not know the logic of `TenantApp` in detail. For example, such `TenantApps` can be written in simply Javascript/Python code running atop bare containers or in work-flows running within, e.g., a Node-RED engine. The provider offers different execution engines like Docker containers or Node-RED. Furthermore, `TenantApp` instances in different edge servers can communicate with each other. The requirements are that they should be able to communicate using services in Edge servers. Here they might face issues related to interoperability w.r.t. protocols and data among servers. Thus, the Edge servers should provide such bridges via the `Edge-to-EdgeConnectivityService`. There could be various reasons for instances of `TenantApps` to exchange data. For example, due to security constraints, one Edge server can be a place where data streams from IoT devices are concentrated whereas another Edge Service is configured to interface to controls of equipment. Therefore, one instance might carry out analytics mainly whilst another instance carries out control commands based on the analytics.

View on IoTaaS and current issues

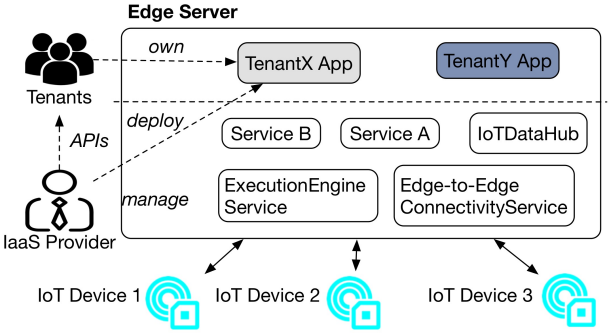

Figure 1. IoT Infrastructure-as-a-Service at the Edge.

Figure 1 shows the normal IoTaaS deployment view, in which tenant apps have direct access to the provided IoTaaS services in the edge; tenant apps use typical user authentication and

[1] `http://www.inter-iot-project.eu/`

service access keys for accessing these services. As IoTaaS is still in its infancy, approaches for security control of tenant apps are needed to fill the gaps in the current IoT platforms and Edge server deployment. For example, MQTT does not provide support for access control management [13] and tenant app's resource consumption control in the Edge. In the current way of dealing with security, such as capability-based control, authentication and authorization would mainly be using "trusts" based on signed paper contracts and the above-mentioned typical service access methods. In the next section, we will present our approach to address these issues.

OVERVIEW OF OUR APPROACH

Key principles of MDSIoT

Having considered the issues discussed in the previous section, we propose an approach to fully govern tenant apps in the Edge. We introduce an intermediate layer to intercept requests using the gatekeeper patterns, plus monitor and use throttling patterns [9], for controlling tenant apps. Specifically, we generate a gatekeeper for each tenant app and deploy both the gatekeeper and the tenant app together in the Edge server. A gatekeeper acts as a proxy for its tenant app to execute and access the base IoT services, according to an execution policy agreed among the tenant and the provider. We leverage the concept of "context" in usage control [17] for flexible access control of IoT tenant apps. There are three main engineering principles for our approach:

1) Increase interoperability support for different tenant apps. We understand that tenant apps can be very different from one to another, such as written in different languages and using different application-level communication protocols. By generating a specific gatekeeper for each tenant app, not only we can control its access to the IoTaaS services but also abstract the communication protocols between tenant apps and the IoTaaS services. In other words, tenant apps are loosely coupled, communication protocols-independent from the IoTaaS services. Generated gatekeepers and IoTaaS platform support for different communication protocols.

2) Distributed, high-performance access control to avoid "bottleneck" of the centralized PDP-PEP model [16], where all access requests from tenant apps via Policy Enforcement Points (PEP) need to be decided by a centralized Policy Decision Point (PDP). This is also important in the IoT scenario as the access might involve voluminous data on the move. Here, with the gatekeeper, we have a specific PDP-PEP for each tenant. This is a lightweight enforcement solution that is especially beneficial for Edge servers, which often have limited resources. If the policy changes, the gatekeeper can be updated at runtime to enforce the latest policy.

3) Flexible monitoring and throttling processes of tenant's activities if over preset limits. Performed as a proxy between a tenant and the provided services, the gatekeeper can detach its tenant from using resources and accessing provided services.

Phases and detailed steps

Figure 2 envisions the typical interaction flows of our approach between various components and security as an output. First, an IoTaaS provider makes IoTaaS APIs publicly available to potential tenants, who develop their IoT apps to leverage the IoTaaS services for their businesses. Tenants are also aware of different (price) models of execution policies set by the provider for tenant apps. Tenants can choose a configuration of execution policy for their app(s) using the `xPolicyTemplate`. The output of this process is the `xPolicyInstance` for a specific tenant that indicates what, when, how, and where IoTaaS services are accessible by the tenant's app(s) as well as the allowed capability for the resources. `xPolicyInstance` is automatically transformed into a platform-specific `Tenant Gatekeeper` component, which is deployed together with the tenant's app(s) in the Edge server to control those app(s).

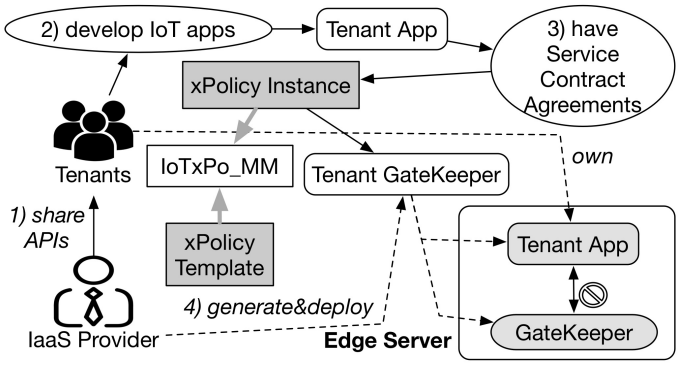

Figure 2. Workflow for specifying and enforcing execution policy in IoT.

Figure 3 shows an overview of the main steps of our approach mapped into DevOps, and different choices of tools for each step. We aim to support for DevOps, with a focus on IoT/Edge applications, to enable continuous delivery of tenant apps[2].

In the planning phase, the IoTaaS provider makes the APIs of its IoTaaS services and the execution policy (`xPolicy`) template available to potential tenants. This step is supported by a public cloud portal where potential tenants can browse IoTaaS services and subscribe to services with execution policies. Based on the provided `xPolicy` template, a tenant registers its IoT service with the IoTaaS provider by specifying how it uses the IoTaaS services: which services it registers to use, data amount, the resource needed and so on. The `xPolicy` template contains the specification on what available options for tenant apps to use the provided IoTaaS services. This template is, in fact, a model that conforms to the metamodel `IoTxPo_MM` presented in the next section. The metamodel can be created in EMF[3] or using UML profiles [5]. The service contract agreements are stored by IoTaaS Provider's system in the form of model instances (`xPolicy` instances) of the metamodel `IoTxPo_MM`.

In the coding phase, based on the provided APIs, tenants will develop their IoT apps that use the provided IoTaaS services for their businesses. Tenants are free to choose how they develop their tenant apps, e.g., simply standalone Javascript/Python code or part of existing frameworks such as SMOOL[4],

[2] `https://www.enact-project.eu`

[3] `https://www.eclipse.org/modeling/emf/`

[4] `https://bitbucket.org/jasonjxm/smool/wiki/Home`

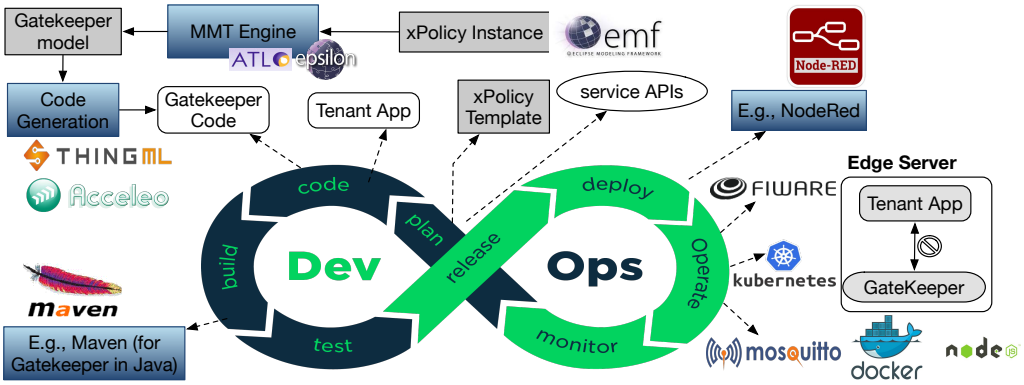

Figure 3. Overview of phases and tool chain for our MDSIoT approach from DevOps viewpoint.

FIWARE[5], or SOFIA2[6]. By using a model-to-model transformation (MMT) engine, such as ATL or Epsilon, we can automatically derive the Gatekeeper ThingML model (code) from the xPolicyInstance. Depending on the target platform for the tenant app, ThingML toolchain generates platform-specific gatekeeper code to be built and deployed together with the tenant app. This step can also be achieved by other model-based code generation tools, such as Acceleo[7], but we chose ThingML toolchain[8] for implementing our prototype because of its specialized support for IoT engineering. We present more about ThingML in the related work section.

In the building phase, depending on the target platform for running tenant apps, different tools can be used, e.g., Maven for Java. We do not focus on **build**, **test**, **release** in this paper.

In the deployment phase, depending on the target platform of the tenant app, we can use different tools for supporting this step such as Docker, Kubernetes[9] or Node-RED[10]. The deployment stage is overlapping with the operation stage in terms of the tool because the same tool can be used in operating the tenant apps and gatekeepers.

In the operation phase, tenant apps can access to the IoTaaS services via the gatekeepers according to execution policies. The gatekeepers are components that not only control tenant apps but also support for tenant apps to access to the IoTaaS services without knowing about the underlying communication protocols. We present more details about this aspect in the next sections.

THE KEY PARTS OF OUR APPROACH

Our approach aims at enabling security control and management at a high level (models) and automating the policy enforcement process from models (e.g., in the cloud or edge server) to code (in edge server, IoT gateway). In other words, our approach is to follow the end-to-end MDE for IoT and along MDE we address execution policy. Figure 4 illustrates the overview of our approach to derive the gatekeeper code from the policy, which briefly described in the next subsection.

[5]`https://www.fiware.org`
[6]`http://sofia2.com/home_en.html`
[7]`http://www.eclipse.org/acceleo`
[8]`https://github.com/HEADS-project/training`
[9]`https://kubernetes.io/`
[10]`https://nodered.org/`

Next, we present our domain-specific language (DSL, namely IoTxPo_MM) for specifying the execution policies of tenant apps. Then, we describe the enforcement framework.

MDS for IoT policy enforcement

In this subsection, we present our MDSIoT approach for specifying and enforcing execution policies for tenant apps in the Edge. Figure 4 shows how execution policies specified at the model level can be enforced at the code level.

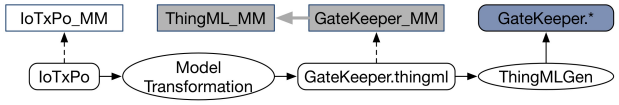

Figure 4. From policy model to gatekeeper code.

On the one hand, we have developed a DSL (IoTxPo metamodel) for specifying IoT execution policy models (IoTxPo models) (see the next section). On the other hand, we use another DSL extended from ThingML [8] to express the bindings between the endpoints of tenant app(s) with the IoTaaS `ServiceEndpoint(s)` and the execution policy for the bindings. The bindings are like port-binding in component-based approaches, which also contain the logic of execution policies. By using model transformations, we transform the IoTxPo model into `GateKeeper.thingml`, which conforms to the ThingML metamodel. The Gatekeepers are ThingML models that can be used to automatically generate platform-specific code for policy enforcement (GateKeeper.*), e.g., Javascript/NodeJS, C or Java. Depending on the platform-specific IoT Gateway and tenant apps, from ThingML models of gatekeepers, we use the ThingML toolchain to generate the gatekeeper code for policy enforcement, which can be deployed together with the corresponding tenant app(s). In this way, IoT tenant apps and gatekeepers can be deployed to enforce execution policies, and even can be updated at runtime. Tenant apps and gatekeepers are organized in a component-based approach to facilitate separation of concerns and reusability. The gatekeepers play as proxy-components to control access and execution between tenant apps and the IoTaaS services.

A metamodel of execution policy for tenants

An IoTaaS provider manages its execution policies for different tenants based on the IoTxPo model, which conforms to

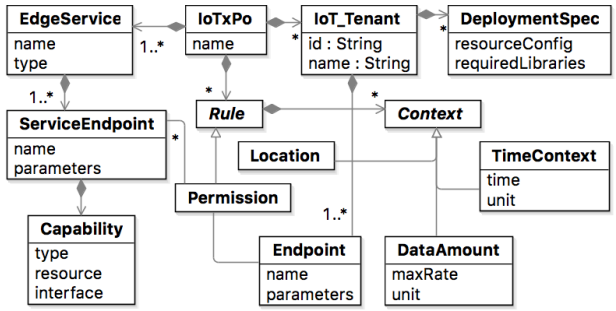

Figure 5. The metamodel of IoT Execution Policy.

the metamodel described in Figure 5. The metamodel shows the conceptual design of our DSL for specifying execution policies of tenant apps. The main concepts of the DSL show what `EdgeService(s)` that the IoT Edge server offers, via what `ServiceEndpoint(s)`. The policy also indicates the `Context` of each `Permission` that a tenant can do with the registered `ServiceEndpoint(s)` as well as when, how it can use these services. Each permission contains the `Capability` of each `ServiceEndpoint` that tenant's `Endpoint` is allowed to use. To support the deployment process, the policy also contains the specification for deployment `DeploymentSpec` for each IoT tenant, such as configuration (`resourceConfig`) and required libraries (`requiredLib`).

As an example, the IoTaaS provider specifies the IoTaaS services (EdgeService) that are deployed on its Edge server(s) such as Security Camera, Light Service, Temperature Sensor, and Presence Sensor. Each service is offered to its tenants via its *ServiceEndpoint* (API) such as request-for-sensor information or command, with the corresponding capability of the service. On the other hand, the policy model also allows specifying each IoT tenant that contracted to use the IoTaaS. It captures the deployment specification(s) of IoT tenant service, e.g., SMOOL libraries, and more importantly, the endpoints that accessing the IoTaaS services. The main part of the policy models specifies the rules about what, when, how tenant apps can use the provided base services. More specifically, each endpoint of IoT tenant is associated with a permission stating which IoTaaS services it has access to, and how it is allowed to use those services, specified in the contexts associated with the rules. For example, a tenant may be only allowed to access to a provided service during a certain time period, with a specific location, and with limitation in reading sensors' data.

IoT execution policy enforcement

This subsection presents the architecture for IoT execution policy enforcement. Figure 6 shows the distributed model for policy enforcement. Each tenant app is deployed with its own local PDP-PEP embedded in the gatekeeper for access control and execution management. Gatekeepers abstract the access from tenant apps to services. Tenant apps cannot access directly to the IoTaaS services without the gatekeepers. Moreover, the PDP in the gatekeeper is fed by the ExecutionEngineService about the real-time contexts of the tenant apps at runtime. This means that the access control decision is not only based on the static policy but also dynamic runtime

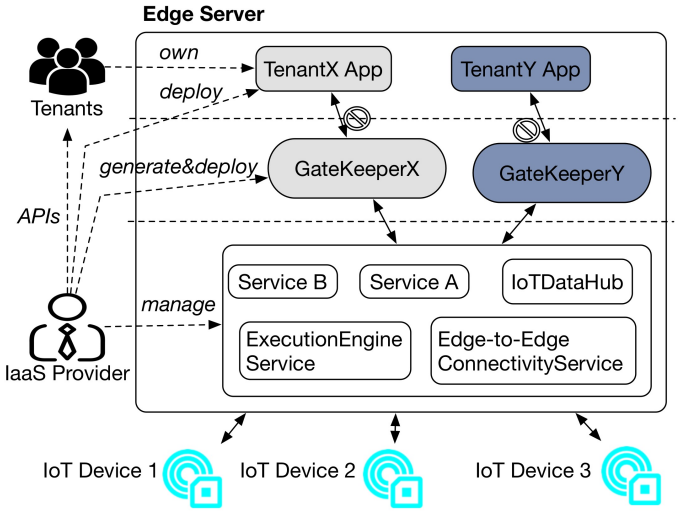

Figure 6. The IoT Execution Policy Enforcement in the Edge.

context, such as a change of location, need for IoT resource balancing, time-factors. In this work, we mainly focus on the access control of tenant apps to the base services. Some simple examples of execution policy instances can be found in [19]. We have not detailed the monitoring and throttling solutions yet. But the basic ideas were to leverage existing execution platform's features such as from Docker or Node-RED to get the runtime status of tenant apps and control them according to execution policies, e.g., throttling [9].

PROTOTYPE FOR A LIGHTWEIGHT IOT EDGE SERVER

Let us consider the case to provision Edge servers with a lightweight configuration, e.g., a typical small server. A simple tenant app to demonstrate for this case is a lighting control app that accesses the IoTaaS services to turn on, turn off, and adjust the lighting in the seaport based on trucks access to lanes and the light sensor's value. The light sensors and events about trucks access to the port are published via MQTT.

Systems view without MDSIoT

Without MDSIoT, because the Edge server configuration is light, we could deploy only basic IoT services, such as MQTT for IoTDataHub, Node-RED for ExecutionEngine, ufw-based firewall[11] for network function and n2disk for DPI. All of such IoT services can be provisioned through a provider. In our test, we use the GenericIoTFunctionProvider[12] to make such a deployment of IoT services for lightweight Edge servers. In this example, since our selected edge server is very lightweight, it is possible that *multiple tenants have to share the same deployment of IoTaaS services and access them (at will)*. The tenant app in the example will need to implement the support for exchanging data using two different communication protocols: MQTT for monitoring data and REST for controlling lights. Each tenant app has to implement its own support for connecting to the services via the provided protocols.

[11] `https://wiki.ubuntu.com/UncomplicatedFirewall`

[12] `https://github.com/rdsea/IoTCloudSamples/tree/master/IoTProviders/GenericLightweightIoTProvider`

MDS-enabled view with MDSIoT

With MDSIoT, we can generate the gatekeepers for different tenant apps accessing different types of IoTaaS services. The deployment process will not only deploy the tenant app but also its gatekeeper, including the support for communication protocols to different IoTaaS services. Figure 7 shows the architecture after deployment. The `TenantApp` is paired with the generated `Adapter4Tenant` via endpoints (like ports in component-based engineering). Listing 1 shows an example of adapter in ThingML that will access to the `ServiceEndpoint` via `Gatekeeper` to provide light sensor's value to `TenantApp`. The `Gatekeeper` contains policy enforcement logic for this `TenantApp`, and only allows data access if the request conforms to the policy for this tenant at runtime (Listing 2). The `ServiceEndpoint` provides the data access layer to IoT-DataHub (like Mosquitto, Listing 3). The `ServiceEndpoint` does not need to be generated but can be already provided by the IoTaaS provider. The `SmoolLightingSensorService` streams light sensor's value to a Smool consumer for calculation, and by combining with truck access data, and take actions like turn on or turn off lights along the lanes. Note that ThingML supports code generation for different target languages such as Java, C, Javascript/NodeJS, or Go, enabling the engineering of our approach for diverse types of Edge servers and IoT services. In this prototype, we use ThingML to generate Java code of gatekeeper (and adapter) for `TenantApp` SmoolLightingSensorService.java.

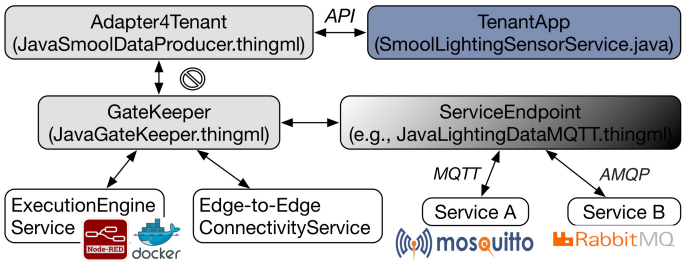

Figure 7. TenantApp accessing ServiceEndpoint via Gatekeeper.

```
import "Protocol.thingml" //common exchanged messages

thing JavaSmoolDataProducer includes Protocol

@src "../src"
@maven_dep "<dependency><groupId>org.smool.kpi</groupId><
    artifactId>common</artifactId><version>2.2.4</version
    ></dependency>"
@maven_dep ...

{
  provided port pp
  @sync_send "true"
  {
    receives lightingSensorData
  }

  statechart SmoolDataProducer init sendingSmoolData {
    state sendingSmoolData {
      internal event eData : pp?lightingSensorData
      action do `
          SmoolLightingSensorProducerLib producer =
    SmoolLightingSensorProducerLib.getInstance();
          if(producer.connect()) {
            producer.sendValue(` & eData.sensorData & `);
          }else {
            System.out.println("\n Producer is not
    connected to SMOOL server!");
          }
      `
    end
  }
}
```

Listing 1. An example of adapter code in ThingML

```
import "Protocol.thingml"

thing JavaGatekeeper includes Protocol
{
  required port rp  {...}
  provided port pp  {...}

  statechart PolicyEnforcer init checkingPolicy {
    on entry print "PolicyEnforcer checking policy!\n"
    state checkingPolicy {
      internal event eData : pp?lightingSensorData
      guard eData.sensorData > 0
      action do
        //checking the policy before sending sensor data
        `if( PDP.isAllowed(`& eData.sensorData &`) ) {`
          rp!lightingSensorData(eData.sensorData)
        `}else{`
          //not allowed
        `};`
    end
          ...
  }
}
```

Listing 2. An example of PolicyEnforcer of JavaGateKeeper.thingml

```
import "Protocol.thingml"
import "JavaSmoolDataProducer.thingml"
import "JavaGatekeeper.thingml"

thing JavaEndpoint includes Protocol {
  provided port pp {...}
  required port rp  {...}

  statechart init INIT {
    state INIT {
      on entry print "MQTTDataService ready! \n"
      internal event eData : pp?lightingSensorData
      rp!lightingSensorData(eData.sensorData)
    }
        ...
  }
}

//TODO: Compile and run for the Java platform
protocol mqtt
@serializer "JSON"
@mqtt_broker_address "localhost" //"test.mosquitto.org"
@mqtt_broker_port "1883"
@mqtt_subscribe_topic "lightingSensor1_Data"
@mqtt_publish_topic "Cmd"

configuration javacfg {
  instance endpoint : JavaEndpoint
  instance smoolDataProducer : JavaSmoolDataProducer
  instance gatekeeper : JavaGatekeeper

  connector endpoint.pp over mqtt
  connector endpoint.rp => gatekeeper.pp
  connector gatekeeper.rp => smoolDataProducer.pp
}
```

Listing 3. A ServiceEndpoint JavaLightingDataMQTT.thingml

RELATED WORK AND DISCUSSION

Model-driven security (MDS) empowers security for software systems by specifying and engineering security requirements with system models [2]. These specifications will be generated into system architectures automatically by corresponding

tools. In the last few years, MDS has been widely researched and applied [15]. In the domain of cyber-physical systems (CPSs), there have been various studies applying MDS in CPSs, however, most of these studies only focus on general security analyses, and thus lacking the engineering security solutions as well as limited tool support [14].

Various application architectures have been proposed for IoT [23–25, 27]. In this work, we consider the IoT infrastructures with edge/fog computing models [4, 21]. In these models, multiple tenant apps can be deployed and executed atop of a fog software infrastructure that provides services to the applications [3]. Examples of this architecture include e.g., Cisco Edge Fog Fabric[13], and Azure IoT Edge[14]. Although there have been several security solutions for IoT, e.g., [6, 10, 18, 20, 22], these approaches mostly focus on the authentication and secure connection among the Things, thus lacking the security for IoT tenant apps in the Edge server with their complexity of the interoperability. Indeed, in [11], the authors have addressed various security requirements for a fog computing architecture for IoT. Despite there are a few general approaches suggested in [11], these have not been realized and are not specific to IoT tenant apps as we consider in this paper.

Standard policy languages such as WS-SecurityPolicy, Role-Based Access Control, or Attribute-Based Access Control, support coarse-grained policies, thus cannot address the policies based on context as we have investigated in this work. There have been a few context-based security policy languages such as aspect-oriented policy [7], ConSpec [1]. However, these languages and related tools can enforce are platform-specific and do not support IoT infrastructure-as-a-service.

Recently, there have been several proposals introducing security policy enforcement for IoT. For example, in [13], the authors presented a model-based toolkit to enforce expressive security policy rules, however, only at the MQTT broker level. Closely related to our work, P4SINC [19] is an execution policy framework that can enforce various generic execution and security policies for IoT application in the Edge. However, the approach in P4SINC is to instrument the IoT applications to inline the policy code into the applications before deploying the applications. Similar to our work's motivation, Velox [26] provided a virtual execution environment can control the execution of IoT applications to ensure their safety and security. Velox introduced a new high-level programming language that can be executed in the virtual environment. This approach requires the IoT applications developed in that specific language, while in our approach, IoT applications can be developed in any language.

As mentioned earlier, we adopt the ThingML [8], proposed by The HEADS FP7 EU project, and leverage its strengths in our implementation. ThingML is a practical MDE approach with a DSL, a toolchain that supports engineering distributed resource-constrained embedded systems, especially IoT systems. ThingML provides methods and tool support for facilitating the integration of resource-constrained embedded systems with more powerful computing resources such as servers and cloud. In other words, ThingML facilitates the collaboration between IoT service developers and IoT infrastructure/platform operators in a large range of processing nodes and protocols with high heterogeneity [12]. One of the key features is ThingML's template mechanism integrated into the language to integrate with third-party (or legacy) APIs, rather than redeveloping them from scratch. This makes ThingML practical for integrating with different existing IoT services, supporting many kinds of target platforms such as Java, NodeJS, Arduino.

[13]Cisco Edge Fog Fabric Data Sheet `https://bit.ly/2LrctgZ`

[14]Azure IoT Edge `https://bit.ly/2JUjPE0`

CONCLUSIONS AND FUTURE WORK

Enforcing execution policies for IoTaaS is challenging. Model-driven engineering approaches enable us to deal with the diversity and complexity of IoT services and tenants in IoTaaS while allowing extensibility and interoperability in analytics and controls of IoT devices across edge nodes in IoTaaS platforms. In this paper, we have introduced an MDS framework called MDSIoT for specifying execution policies of IoTaaS tenants and enforcing the policies in the Edge. MDSIoT consists of a DSL for specifying execution policies and a chain of model transformations and code generation to generate platform-specific gatekeepers from policies. The generated gatekeepers, which are deployed together with tenant applications, act as an intermediate lightweight proxy layer to control the access of tenant applications to the IoTaaS services of the IoTaaS provider. We map the main steps of our framework into the steps and possible tools for DevOps of IoTaaS, because we want to support DevOps for IoTaaS engineering. We have demonstrated our approach in a prototype implemented using ThingML, a practical MDE approach with tool support specialized for IoT engineering.

Our work is still at an early stage with the focus on architectural designs, policy modeling, and engineering approach. So far, we have mainly addressed access control of IoTaaS tenant apps. We will improve our framework with the details of monitoring and throttling techniques to better control tenant apps. Another step is to work on the implementation of tools and to perform various Edge analytics and control experiments with real IoT services in telco and logistics domains to predict maintenance and seaport management.

Acknowledgements This work is partially supported by the European Community's H2020 programme under grant agreement no 780351 (ENACT), the H2020 INTER-IoT through the subproject INTER-HINC, and the University of Dayton Research Council Seed Grant. We would like to thank Dr. Brice Morin for his help with ThingML.

REFERENCES

1. Irem Aktug and Katsiaryna Naliuka. 2008. ConSpec – A Formal Language for Policy Specification. *Sci. Comput. Program.* 74, 1-2 (Dec. 2008), 2–12.
2. David Basin, Jürgen Doser, and Torsten Lodderstedt. 2006. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2006).
3. Charles C Byers and Patrick Wetterwald. 2015. Fog computing distributing data and intelligence for resiliency

and scale necessary for IoT: The Internet of Things (ubiquity symposium). *Ubiquity* 2015, November (2015).

4. Mung Chiang and Tao Zhang. 2016. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal* 3, 6 (2016), 854–864.
5. Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. 2004. An introduction to UML profiles. *UML and Model Engineering* 2 (2004).
6. Pascal Gremaud, Arnaud Durand, and Jacques Pasquier. 2017. A secure, privacy-preserving IoT middleware using intel SGX. In *Proceedings of the Seventh International Conference on the Internet of Things*. ACM, 22.
7. Kevin W. Hamlen and Micah Jones. 2008. Aspect-oriented In-lined Reference Monitors. In *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS '08)*. ACM, New York, NY, USA, 11–20.
8. Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. ThingML: A Language and Code Generation Framework for Heterogeneous Targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)*. ACM, New York, NY, USA.
9. Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, and Trent Swanson. 2014. *Cloud design patterns: Prescriptive architecture guidance for cloud applications*. Microsoft patterns & practices.
10. Parikshit N Mahalle, Bayu Anggorojati, Neeli R Prasad, Ramjee Prasad, and others. 2013. Identity Authentication and Capability Based Access Control (IACAC) for the Internet of Things. *Journal of Cyber Security and Mobility* 1, 4 (2013), 309–348.
11. Bridget A Martin, Frank Michaud, Don Banks, Arsalan Mosenia, Riaz Zolfonoon, Susanto Irwan, Sven Schrecker, and John K Zao. 2017. OpenFog security requirements and approaches. In *Proceedings of Fog World Congress (FWC), 2017 IEEE*. IEEE, 1–6.
12. Brice Morin, Nicolas Harrand, and Franck Fleurey. 2017. Model-based software engineering to tame the iot jungle. *IEEE Software* 34, 1 (2017), 30–36.
13. Ricardo Neisse, Gary Steri, and Gianmarco Baldini. 2014. Enforcement of security policy rules for the Internet of Things. In *Proceedings of the 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 165–172.
14. Phu H. Nguyen, Shaukat Ali, and Tao Yue. 2017. Model-based security engineering for cyber-physical systems: A systematic mapping study. *Information and Software Technology* 83 (2017), 116 – 135.
15. Phu H. Nguyen, Max Kramer, Jacques Klein, and Yves Le Traon. 2015. An extensive systematic review on the Model-Driven Development of secure systems. *Information and Software Technology* 68 (2015), 62 – 81.
16. Phu H Nguyen, Gregory Nain, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. 2014. Modularity and dynamic adaptation of flexibly secure systems: Model-driven adaptive delegation in access control management. In *Transactions on Aspect-Oriented Software Development XI*. Springer, 109–144.
17. Jaehong Park and Ravi Sandhu. 2004. The UCON ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)* 7, 1 (2004), 128–174.
18. Ankush B Pawar and Shashikant Ghumbre. 2016. A survey on IoT applications, security challenges and counter measures. In *Proceedings of the International Conference on Computing, Analytics and Security Trends (CAST)*. IEEE, 294–299.
19. Phu H. Phung, Hong-Linh Truong, and Divya Teja Yasoju. 2017. P4SINC-An Execution Policy Framework for IoT Services in the Edge. In *Proceedings of the 2017 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 137–142.
20. Wissam Razouk, Daniele Sgandurra, and Kouichi Sakurai. 2017. A new security middleware architecture based on fog computing and cloud to support IoT constrained devices. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. ACM, 35.
21. Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.
22. M. Singh, M. A. Rajan, V. L. Shivraj, and P. Balamuralidhar. 2015. Secure MQTT for Internet of Things (IoT). In *Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies*. 746–751.
23. K Thramboulidis, P Bochalis, and J Bouloumpasis. 2017. A framework for MDE of IoT-based manufacturing cyber-physical systems. In *Proceedings of the Seventh International Conference on the Internet of Things*. ACM.
24. Aparna Saisree Thuluva, Arne Bröring, Ganindu P Medagoda Hettige, Darko Anicic Don, and Jan Seeger. 2017. Recipes for IoT applications. In *Proceedings of the Seventh International Conference on the Internet of Things*. ACM, 10.
25. Hong-Linh Truong and Schahram Dustdar. 2015. Principles for engineering IoT cloud systems. *IEEE Cloud Computing* 2, 2 (2015), 68–76.
26. Nicolas Tsiftes and Thiemo Voigt. 2018. Velox VM: A safe execution environment for resource-constrained IoT applications. *Journal of Network and Computer Applications* (2018).
27. Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. 2017. Internet of Things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE wireless communications* 24, 3 (2017), 10–16.