Towards Model-Based Communication Control for the Internet of Things

Imad Berrouyne, Mehdi Adda, Jean-Marie Mottu, Jean-Claude Royer, Massimo Tisi

Towards Model-Based Communication Control for the Internet of Things

Imad Berrouyne[1], Mehdi Adda[2], Jean-Marie Mottu[1], Jean-Claude Royer[1], and Massimo Tisi[1]

[1] Naomod Team, IMT Atlantique, LS2N, Nantes, France
{firstname.lastname}@ls2n.fr
[2] Mathematics, Computer Science and Engineering Dep. University of Quebec At Rimouski, Rimouski, QC G5L 3A1, Canada
{firstname.lastname}@uqar.ca

Abstract. Most of existing Model-Driven Engineering (MDE) approaches for the Internet of Things (IoT) focus on means of modeling the behavior of end devices. Little attention has been paid to network-related abstractions and communication control. The paper introduces an approach towards enabling model-based communication control in a network of things. First, we suggest a Domain Specific Language (DSL) to abstract basic network features. Second, we propose a policy language to control the communications within the network. Finally, as a proof-of-concept, we present a code generation process to enforce the expressed policy at runtime.

Keywords: Internet of Things · Model-Driven Engineering · Networking · Publish/Subscribe · Communication Control.

1 Introduction

The IoT is reshaping our society's relationship with information and technology. Gartner reports that more than 8 billion connected devices are in use, and forecasts that this number will grow to 20.4 billion by 2020 [11]. Communication is the backbone of the IoT, which consists of connecting various computational platforms ranging from tiny and resource-constrained sensors and actuators to smartphones and computers.

In the light of recent large-scale network attacks such as Mirai and Persirai [29,31,16] targeting numerous devices, the need of new security approaches with respect to communication has resurfaced. As a matter of fact, existing engineering models for the IoT have shown their limits w.r.t security [19,27]. Indeed, according to the SANS Institute, almost 90% of security professionals affirm that changes to security controls are required when it comes to the IoT [24].

Most of these approaches are rather time-consuming and require learning platform specificity in detail as well as expertise in order to build efficient and secure IoT applications. Because of these difficulties, buggy and insecure IoT applications may easily be delivered [14,27].

MDE is an emerging and promising paradigm having the potential to overcome such issues (e.g., platforms heterogeneity, inconsistent security specification). All the more so that recently MDE has successfully been applied to adaptive and distributed systems, by the model@runtime approach [2] as well as in model-driven security [1,20]. MDE can help in designing correct communications and secure systems by abstracting network and security features. Then, by means of code generation tools, guarantee that properties are enforced at runtime. Furthermore, it also allows for reasoning formally on models for various purposes such as security analysis and threat assessment [21], to name just a few. However, although the ongoing work on abstracting device heterogeneity is rather significant [13,10,3,4], modeling then enforcing security policies in the IoT is understudied.

This paper represents a first step towards a MDE approach focusing on communication control in a network of distributed things. Our approach relies on the abstract description of the network configuration and its communication policy as well as a code generation process for enforcement at runtime.

The paper is structured as follows. Section 2 presents a running example of IoT system. Section 3 gives an overview of the existing works. Section 4 provides our concrete solution based on a DSL and a code generation procedure. Finally, Section 5 presents the conclusion and future work.

2 Running case

Fig. 1 depicts an overall view of a small running case, including the used material as well as the possible interactions. We consider two rooms, each one containing the following things: a Temperature Sensor (TS) and a Smart Air Conditioner (SAC). A user monitors the temperature in both rooms, using a mobile interface. From a technical perspective, we use Arduino boards for the sensors and actuators in the network.

As customary in IoT, communication between things is ensured by a Publish and Subscribe (PubSub) channel. A thing publishes its data to a topic, then another thing can consume this data by subscribing to this topic. On the one hand, the TSs collect the current temperature in the room and publish it to a given topic in the broker. On the other hand, each SAC subscribes to the temperature measurements of the room it is located in, in order to decide how to behave. The monitor receives data from all devices and shows it on its screen and commands remotely the SACs. Concretely, Message Queuing Telemetry Transport (MQTT) is used as a PubSub communication channel and Mosquitto [17] as an open-source MQTT PubSub broker. MQTT is a popular communication protocol [5,18] to build applications where things need to collaborate towards a common goal.

Fig. 2 depicts the internal behavior of the TS and SAC using statecharts. In particular, the TS statechart's state `SendTemperature` executes the action of sending measurements through a channel. On the other side, the SAC state-

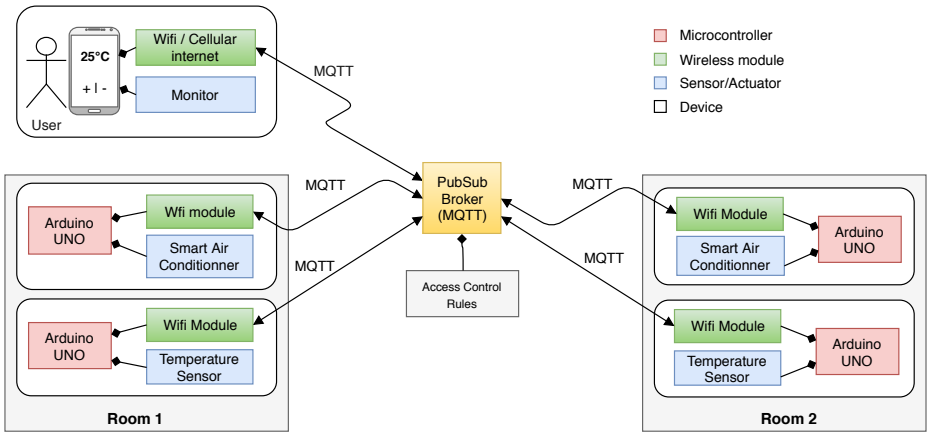

Fig. 1. Overview of the running case

chart's state `WaitTemperature` waits for the temperature measurements from the channel to adjust its behavior.

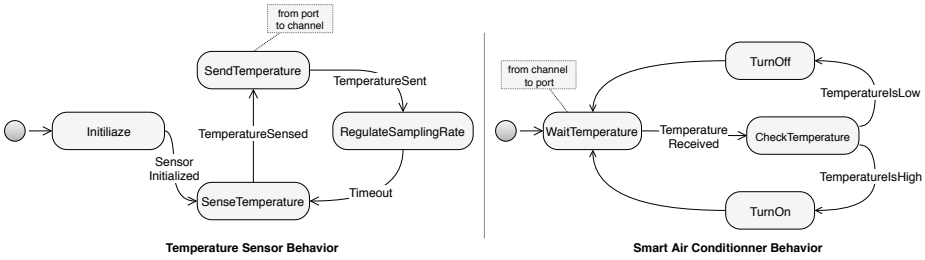

Fig. 2. Behavior of things in the running case

State-of-art MDE tools for IoT allow developers to define such statecharts and generate code from them. In particular, ThingML [13] is a DSL used to model the things behavior as communicating event-based statecharts, typically encapsulating platform-specific code (e.g., C code on Arduino).

To ensure secure interactions among things we need to control the communications using a network policy. A policy is a flexible means to secure a network, through various control points. Controls can be enforced in the channel or directly in the devices. In particular, things cannot communicate between each other unless they are authorized to do so. For example, we have to ensure that the SAC from room 1 can only access the temperature data of room 1, and is denied access to data of room 2. Moreover, to avoid an unexpected behavior in the SAC, we use the policy to allow for communication between the SAC and the sensors only when the temperature is within a given range. For example the

SAC can receive temperature measurements only when it is between -20°C and 50°C.

Existing MDE approaches like ThingML do not include a model of the network, thus hampering global reasoning on the network behavior, and lack mechanisms to model and enforce security measures.

3 Related work

ThingML [13] proposes a methodology for the IoT using established MDE techniques [23]. The language has shown its efficiency at abstracting hardware and programming languages [22,30]. The approach provides a DSL to design the things' internal behavior using statecharts and an extensible multi-platform code generation framework. The latter also provides a plug-in system to add a network client to things. However, abstractions w.r.t communication are rather minimal in the DSL, simply consisting in declaring the used protocol and its attributes.

Eclipse Vorto [10] provides a solution to abstract the device capabilities into functions. A function consists of a set of attributes and a set of operations using the attributes. The functions are grouped inside a model to describe the behavior of the device. Code generators for various platforms permit to produce code from this model. The solution also offers a repository to share and reuse models and code generators. Compared to ThingML, modeling the device behavior is limited, only few operations are achievable. Communication is not modeled.

SensIDL [26] provides a MDE approach to tackle the data format heterogeneity among IoT devices. Indeed, a developer describes a platform-agnostic representation of the data generated by devices. Then, a multi-platform code generator produces the communication interfaces as well as the mechanisms to encode and decode this data on every device. Abstraction w.r.t to the network as well as security are not covered. In addition, contrary to ThingML, modeling of the device behavior is not included in the process.

Most of the existing MDE approaches that address network-related modeling, target Wireless Sensor and Actuators Networks (WSAN). For instance in [9], the authors map the Specification and Description Language (SDL) with TinyOS component models to enable a formal description of communication protocols. Then, a general scheme for creating code from these models is proposed.

From a security perspective, Basin et. al [1] present a comprehensive overview of model-driven security approaches. The considered works allow for modeling security requirements along with the system design, and generate security mechanisms at runtime. The authors show, using a concrete example, how a security policy model is transformed by a code generation tool to control the behavior of a Graphical User Interface (GUI) at runtime. However, distributed systems such as the IoT and platform heterogeneity are not considered.

In [21], Mavropoulos et al. suggest a metamodel to describe IoT systems along with their security aspects. In this respect, a DSL is used to abstract hardware, software, social and security concepts. The approach is not meant for code generation, but rather for security analysis and visualization.

The OASIS consortium provides a framework to express and enforce communication policies. It defines a language called eXtensible Access Control Markup Language (XACML) to express an Access Control (AC) policy in Extensible Markup Language (XML) format [6]. It relies on a request-response model, AC decisions are taken dynamically. It also defines the mechanisms to process this policy. The security framework needs systematically a centralized Policy Decision Point (PDP) to evaluate access requests vis-a-vis the policy, while we are interested in distributing the enforcement of the security policy.

Martínez et al. [20] propose a model-driven reverse engineering approach to obtain a Platform-Independent Model (PIM) of the global AC policy in a network. The approach uses the firewalls configuration files in the system to extract all AC rules. Those rules are transformed into PIMs for each firewall then merged into a global AC model. A XACML policy can be easily generated from this model. We plan to provide an integrated modeling language for the IoT, including device behavior and network structure, and a policy language on top of those.

4 Approach

Our objective is providing a model-based methodology to control communications in a network of heterogeneous, distributed and connected devices. In this paper we design the main components of this methodology: a DSL to model IoT networks (Section 4.1), a policy language to control the network communications (Section 4.2), and a code generation process to enforce the expressed policy at various points of the architecture (Section 4.3). Our proposal is built on top of ThingML, from which we reuse the models of the thing behavior and the multi-platform code generator. The language development is open, and source code can be accessed online [3].

Throughout the paper, we use the running case to illustrate different facets of the methodology. For the sake of simplicity, we use a basic authentication mechanism, identifying things by a username and a password. Moreover, considerations on trust are beyond the scope of the paper.

4.1 DSL for network modeling

To express the features of the network we define a textual language whose metamodel is depicted in Fig. 3. We use Xtext [12] to define the grammar of the concrete textual syntax.

A thing's internal behavior is described by importing a ThingML model. Listing 1.1 depicts the declaration of the things in the example. For instance, `import Temperature "temperature.thingml"` imports the model (i.e., statechart) of the TS.

Network communication is abstracted by a concept of **channel**. The current language supports only one type of channels, **channel : pubsub**. We plan to

[3] https://github.com/atlanmod/iotdsl

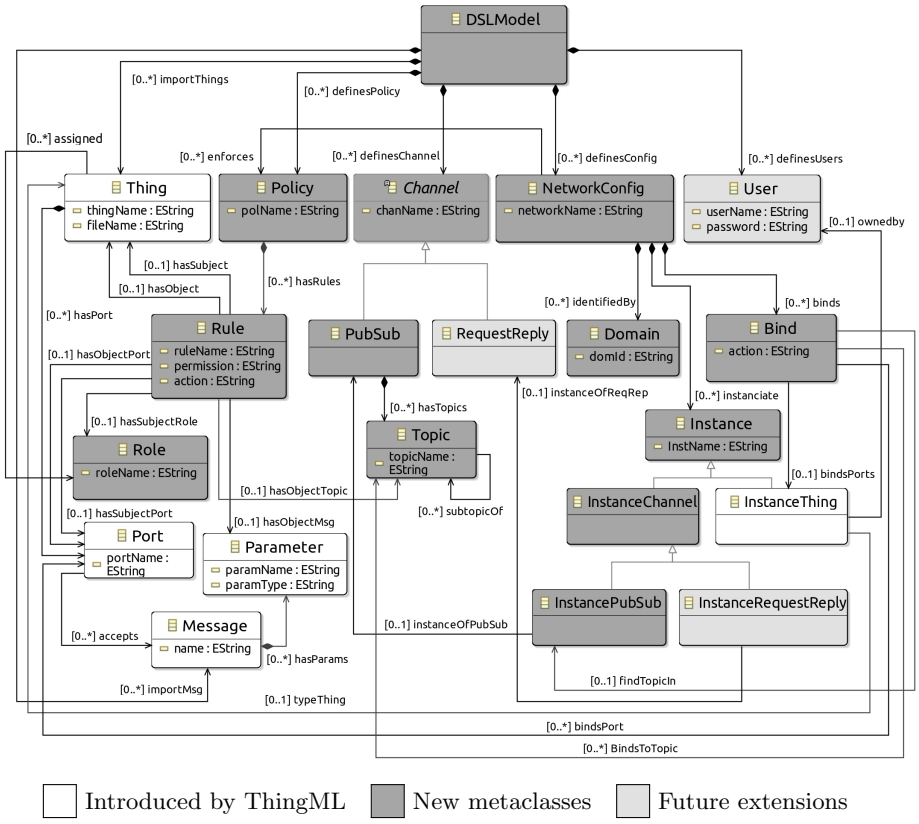

Fig. 3. Metamodel of the DSL

include other types of channel, such as request-reply (e.g., HTTP, CoAP). As it can be seen in Listing 1.2, a PubSub channel may contain multiple topics. The keyword **subtopicOf** provides a basic hierarchical structure for the topics.

The **networkConfig** section describes the global network topology. For instance, it defines which instances of things and channels are available in the network, then it binds a things' ports to channels to create a communication scheme. Listing 1.3 provides a configuration of the running case. A **networkConfig** has a **domain**, that is unique and serves as a global identifier for the network [25]. For instance, in our running case we use the domain in the topic structure as the root topic of the channel. A **bind** declaration connects a thing's port to a PubSub channel, by subscribing or publishing to its topics.

A **networkConfig** can also enforce a policy in the network. Multiple policies can be enforced. For instance, in Listing 1.3, both `roleBasedPolicy` as well as `attributeBasedPolicy` are enforced. Control strategies are discussed further in the next section.

```
1 import Temperature "temperature.thingml" assigned sensor // assigned role sensor
2 import AirConditionner "airconditionner.thingml" assigned actuator
3 import Monitor "monitor.thingml" assigned actuator
```

Listing 1.1. Import of ThingML files

```
1 channel:pubsub MQTTChannel {
2     topic room1 // One topic per room
3     topic room2
4     topic temperatureData1 subtopicOf room1
5     topic commands1 subtopicOf room1
6     topic temperatureData2 subtopicOf room2
7     topic commands2 subtopicOf room2
8 }
```

Listing 1.2. Definition of channels

4.2 Policy language

A policy contains a set of rules. We define a rule as the composition of a subject (Thing, InstanceThing, Port or Role), a permission (allow or deny), an action (send or receive) and an object (Thing, InstanceThing, Port, Message or Topic). At this stage we only cover few mechanisms of the Role-Based Access Control (RBAC) [7] and Attribute-Based Access Control (ABAC) [15] security models. You can find examples of the policy language in Listings 1.4 and 1.5.

As an illustration we apply these control strategies in our running case. The main goal is to avoid any unexpected behavior from the network.

RBAC. This is a coarse-grained strategy consisting of defining roles then assigning them to things. All the permissions given to a role will be applied to all things with that role. This allows to decouple permissions from the concrete development of things.

In our running case we define two roles: one for sensors and one for actuators. The first role gives only send permission to all topics while the second one gives only receive permission from all topics. We assign the sensor role to the TS, the actuator role to the SAC, and both roles to the monitor as it needs to receive the temperature and to send commands to the SAC. Listing 1.4 shows how this is defined.

ABAC. A more fine-grained strategy consists on dynamically deciding to allow the communication, based on contextual attributes. As a proof-of-concept we provide basic ABAC mechanisms.

For instance, in Listing 1.5, the rule in line 2 allows communication based on the source and destination ports (`Temperature.temperaturePort` and `Monitor.temperaturePort`). Lines 4-5 specify that only temperature messages whose

```
networkConfig smarthomeConfiguration {
    domain "fr.naomod.smarthome"
    enforce roleBasedPolicy, attributeBasedPolicy
    ....// Instances declaration
    bind instanceTS1.temperaturePort => MQTTChannel{temperatureData1}
    bind instanceTS2.temperaturePort => MQTTChannel{temperatureData2}
    bind instanceSAC1.temperaturePort <= MQTTChannel{temperatureData1}
    bind instanceSAC2.temperaturePort <= MQTTChannel{temperatureData2}
    bind instanceSAC1.commandsPort <= MQTTChannel{commands1}
    bind instanceSAC2.commandsPort2 <= MQTTChannel{commands2}
    bind instanceMonitor.temperaturePort <= MQTTChannel{temperatureData1,
        temperatureData2}
    bind instanceMonitor.commandsPort => MQTTChannel{commands1}
    bind instanceMonitor.commandsPort2 => MQTTChannel{commands2}
}
```

Listing 1.3. Network configuration (=> publish, <= subscribe)

```
policy roleBasedPolicy {
    rule role:sensor allow:send topic:room1,room2
    rule role:actuator allow:receive topic:room1,room2
}
```

Listing 1.4. Role-Based policy

value is in a certain range, can be communicated. Being able to decide based on the content of the communication, provides a fine granularity for access control.

4.3 Code generation process

Our code generation process is depicted in Fig. 4. The code generator takes as input a file containing the **networkConfig**. It performs two main functions: first it transforms ThingML models to bind them with specific network channels, second it enforces the policy at various enforcement points.

As depicted in Fig. 5, controls are enforced at various points of the network architecture: 1) in the broker, by controlling the access to topics, or 2) in the thing by changing its internal behavior in the model.

For our previous RBAC example, controls are applied only in the broker. In particular, our current generator is able to produce Access Control Rules (ACR) for the Mosquitto, specifying which MQTT topics can be accessed by each thing.

Content-based policies, like in Lines 4-5 of Listing 1.5, cannot usually be implemented in the broker because, for performance reasons, only few brokers provide content-based PubSub [28]. In this case, the control will be performed in the things rather than the broker, during the operations of send and receive. Distributed content-based PubSub has also the advantage to be more scalable

```
1 policy attributeBasedPolicy {
2     rule Temperature.temperaturePort allow:send port:Monitor.temperaturePort
3     rule Temperature.temperaturePort allow:send thing:AirConditionner
4     rule Monitor allow:receive message:temperatureMessage.currentTemperature <
          50 and temperatureMessage.currentTemperature > −20
5     rule Temperature allow:send message:temperatureMessage.currentTemperature
          < 100
6 }
```

Listing 1.5. Attribute-Based policy

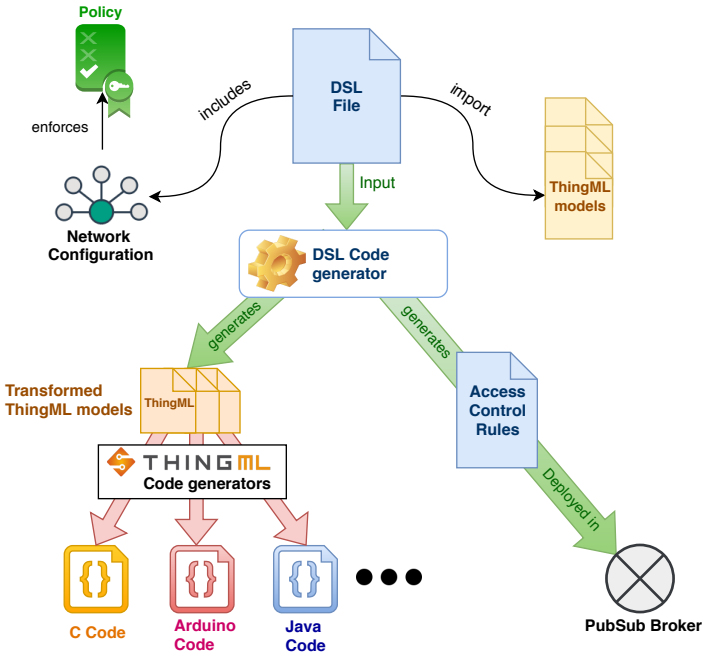

Fig. 4. Code generation procedure

and flexible. It avoids the "single point of failure" risk associated with control on the broker. This also contributes to a better security by design [8] as well as to reduce the attack surface of the thing.

In our language we can decide to control communication on the send or receive. As shown in Fig. 2, the TS sends its data in the `SendTemperature` state. To implement the rule in Line 5 of Listing 1.5 the generator adds an **`if`** condition before performing the send in the `SendTemperature` state, as shown in Listing 1.6. Temperature measurement is sent only when it is lower than 100. Likewise, Fig. 2 also shows that the SAC can receive this data at the `WaitTemperature` state. To control the temperature received by the SAC

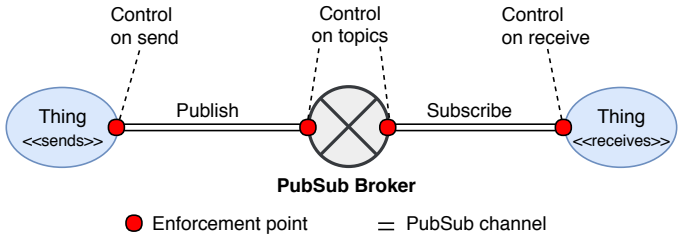

Fig. 5. Enforcement points

```
1 if (currentTemperature<100) { // Added control
2     temperaturePort!temperatureMessage(currentTemperature)
3 }
```

Listing 1.6. Control on send in ThingML

according to the rule in Line 4 of Listing 1.5, the generator adds a guard to the incoming event, as shown in Listing 1.7. Temperature is accepted only when it is between -20 and 50.

Controlling communication on receive requires checking whether the message satisfies the control conditions before reception. The message can still be intercepted by other nodes, and demands superfluous processing from the thing for a message, that probably will not be used. When communication is controlled on send, the message remains in the thing until it satisfies the control conditions, this is more secure as the thing keep control over the message. However a malicious developer could easily remove a control on send, with the objective of controlling another connected device.

5 Conclusion

We extended the current work on MDE for the IoT with a model-based communication control approach. In this respect, we proposed a DSL to tackle the lack of network modeling.

```
1 internal event receivedTemperature : temperaturePort?temperatureMessage
2 guard receivedTemperature.currentTemperature < 50
3     and receivedTemperature.currentTemperature > −20 // Added control
4 action do
5 ... // Actions
6 end
```

Listing 1.7. Control on receive in ThingML

Network-related abstractions are proposed. The study focuses on PubSub communication channels and permit to model a network of things. Communication control is achieved using rule-based policies. The proposed policy language permits to describe basic concepts of established security models such RBAC and ABAC.

A code generation process enforces the policy at various points of the network architecture. In this respect, AC rules are generated to be deployed in the broker and things' internal behavior may be modified.

In future work, we will enrich this first approach with more network-related abstractions. Then, we will formalize the model transformations of ThingML models using the AtlanMod Transformation Language (ATL). Finally, we plan to improve the security mechanisms already in place with smarter controls distribution throughout the enforcement points.

References

1. David Basin, Manuel Clavel, and Marina Egea. A decade of model-driven security. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 1–10. ACM, 2011.
2. Gordon Blair, Nelly Bencomo, and Robert B France. Models@ run. time. *Computer*, 42(10), 2009.
3. Mike Botts and Alexandre Robin. Opengis sensor model language (sensorml) implementation specification. *OpenGIS Implementation Specification OGC*, 2007.
4. Christian Bunse, Hans-Gerhard Gross, and Christian Peper. Applying a model-based approach for embedded system development. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on.* IEEE, 2007.
5. Komkrit Chooruang and Pongpat Mangkalakeeree. Wireless heart rate monitoring system using mqtt. *Procedia Computer Science*, 86:160–163, 2016.
6. OASIS XACML Technical Committee et al. extensible access control markup language (xacml) version 3.0. *Oasis standard, OASIS*, 2013.
7. J Cugini, R Kuhn, and D Ferraiolo. Role-based access control: Features and motivations. In *Proceedings of the Annual Computer Security Applications Conference, Los Alamitos, Calif., 1995*, 1995.
8. Noopur Davis, Watts Humphrey, Samuel T Redwine, Gerlinde Zibulski, and Gary McGraw. Processes for producing secure software. *IEEE Security & Privacy*, 2(3):18–25, 2004.
9. Daniel Dietterle, Jerzy Ryman, Kai Dombrowski, and Rolf Kraemer. Mapping of high-level sdl models to efficient implementations for tinyos. In *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pages 402–406. IEEE, 2004.
10. Eclipse. Eclipse Vorto - IoT Toolset for standardized device descriptions.
11. UK Egham. Gartner says 8.4 billion connected" things" will be in use in 2017, up 31 percent from 2016. *Gartner, Inc*, 7, 2017.
12. Moritz Eysholdt and Heiko Behrens. Xtext: Implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pages 307–309, New York, NY, USA, 2010. ACM.
13. Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 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, pages 125–135. ACM, 2016.
14. Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. Smart nest thermostat: A smart spy in your home. *Black Hat USA*, 2014.
15. Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.
16. Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.
17. Roger A Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13), 2017.
18. Jorge E Luzuriaga, Juan Carlos Cano, Carlos Calafate, Pietro Manzoni, Miguel Perez, and Pablo Boronat. Handling mobility in iot applications using the mqtt protocol. In *Internet Technologies and Applications (ITA), 2015*. IEEE, 2015.
19. Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*, pages 336–341. IEEE, 2015.
20. Salvador Martínez, Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, and Jordi Cabot. Model-Driven Extraction and Analysis of Network Security Policies. In *Model-Driven Engineering Languages and Systems*, pages 52–68, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
21. Orestis Mavropoulos, Haralambos Mouratidis, Andrew Fish, and Emmanouil Panaousis. Asto: A tool for security analysis of iot systems. In *Software Engineering Research, Management and Applications (SERA), 2017 IEEE 15th International Conference on*, pages 395–400. IEEE, 2017.
22. Brice Morin, Nicolas Harrand, and Franck Fleurey. Model-based software engineering to tame the iot jungle. *IEEE Software*, 34(1):30–36, 2017.
23. Jishnu Mukerji and Joaquin Miller. Mda guide version 1.0. 1. *Object Management Group*, 2003.
24. John Pescatore and Gal Shpantzer. Securing the internet of things survey. *SANS Institute*, pages 1–22, 2014.
25. Lauri IW Pesonen, David M Eyers, and Jean Bacon. Access control in decentralised publish/subscribe systems. *JNW*, 2(2):57–67, 2007.
26. Christoph Rathfelder and Emre Taspolatoglu. SensIDL: Towards a generic framework for implementing sensor communication interfaces, 2015.
27. Yogeesh Seralathan, Tae Tom Oh, Suyash Jadhav, Jonathan Myers, Jaehoon Paul Jeong, Young Ho Kim, and Jeong Neyo Kim. Iot security vulnerability: A case study of a web camera. In *Advanced Communication Technology (ICACT), 2018 20th International Conference on*, pages 172–177. IEEE, 2018.
28. Haiying Shen. Content-based publish/subscribe systems. In *Handbook of Peer-to-Peer Networking*, pages 1333–1366. Springer, 2010.
29. Trend Micro. TrendLabs Security Intelligence BlogPersirai: New Internet of Things (IoT) Botnet Targets IP Cameras - TrendLabs Security Intelligence Blog. 2017.
30. Anatoly Vasilevskiy, Brice Morin, Øystein Haugen, and Pal Evensen. Agile development of home automation system with thingml. In *Industrial Informatics (INDIN), 2016 IEEE 14th International Conference on*. IEEE, 2016.
31. Nicky Woolf. Ddos attack that disrupted internet was largest of its kind in history, experts say. *The Guardian*, 26, 2016.