UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA - VARESE

DiSTA

Dipartimento di Scienze Teoriche e Applicate

P H D T H E S I S

to obtain the title of

Doctor of Science

Specialty : Computer Science

Defended by

ENGIN DENIZ TÜMER

ACCESS CONTROL WITHIN MQTT-BASED IOT ENVIRONMENTS

Advisor: PROF. ELENA FERRARI

Advisor: ASSOC. PROF. PIETRO COLOMBO

defended on February XX, 2022

To my precious, wonderful and beloved dad and mom,

who have raised me to be the person I am today and support me every time.

Acknowledgments

Throughout the writing of this thesis, I have received a great deal of support and assistance from my supervisors, I would like to express my appreciation to Dear Prof. Elena Ferrari and Assoc. Prof. Pietro Colombo. I've learned many things from you that affect both my academic and personal life. I will always be thankful to you to give me the motivation to finish this thesis. Also, I would like to express sincere gratitude to Dear Prof. Barbara Carminati for her guidance during my PhD education.

I would like to express my thanks to my thesis reviewers, XXXXX**??**, and YYYYY**??** to spend their precious time reviewing my thesis and giving me an opportunity to improve it based on their instructive and valuable comments.

Finally, I would like to thank you, my beloved colleagues Gökhan Sağırlar, Stefania Boffa, Zulfikar Alom, Christian Rondanini, Anh-Tu Hoang, Federico Daidone, Ha Xuan Son, Ahmed Lekssays, Thanh Loan Nguyen and Giorgia Sirigu. They've helped me to get used to life in Italy and shared enjoyable and precious moments with me. I hope each of you has a great academic and personal life. I also would like to give my thanks to Mauro Santabarbara, and Roberta Viola to help me in the department.

Abstract

IoT applications, which allow devices, companies, and users to join the IoT ecosystems, are growing in popularity since they increase our lifestyle quality day by day [10]. For instance, many IoT applications assist users during their daily routines by using wearable technologies(e.g., sport training or health monitoring). However, due to the personal nature of the managed data, numerous IoT applications represent a potential threat to user privacy and data confidentiality (e.g., see [1]).

Insufficient security protection mechanisms in IoT applications can cause unauthorized users to access data. To solve the security issue related to unauthorized accesses, the access control systems, which guarantee only authorized entities to access the resources, are proposed in both academic and industrial environments. The main purpose of access control systems is to determine who can access specific resources under which circumstances via the access control policies. An access control model encapsulates the defined set of access control policies.

Access control models have been proposed also for IoT environments to protect resources from unauthorized users. The following families of access control models are generally the basis of access control solutions for IoT environments: Capability-Based Access Control(CapBAC) [42], Usage Control(UCON) [66], Role-Based Access Control(RBAC) [74], and Attribute-Based Access Control(ABAC) [48]. Among the solutions based on these access control models, the proposals which are based on Attribute-Based Access Control (ABAC), have been widely adopted in the last years. In the ABAC model, authorizations are determined by evaluating attributes associated with the subject, object, and environmental properties. The subject is granted appropriate access permissions by the system according to his/her attributes when he/she issues an access request. ABAC model provides outstanding flexibility and supports fine-grained, context-based access control policies. These characteristics perfectly fit the IoT environments.

In this thesis, we employ ABAC to regulate the reception and the publishing of messages exchanged within IoT environments. Moreover, we select an ABAC framework proposed in [22], which regulates data sharing within an MQTT-based IoT environment, as the base structure for our thesis. MQTT [12] is a standard application layer protocol that enables the communication of IoT devices by means of the publish/subscribe architecture. The ABAC framework proposed in [22] provides a centralized enforcement mechanism that allows enforcing ABAC policies to handle data sharing within an MQTT-based IoT environment.

Even though the current access control systems tailored for IoT environments in the literature handle data sharing among the IoT devices by employing various access control models and mechanisms to address the challenges that have been faced in IoT environments, surprisingly two research challenges have still not been sufficiently examined.

The first challenge that we want to address in this thesis is to regulate data sharing among interconnected IoT environments. In interconnected IoT environments, data exchange is carried out by devices connected to different environments. The majority of proposed access control frameworks in the literature aimed at regulating the access to data generated and exchanged within a single IoT environment by adopting centralized enforcement mechanisms. However, currently, most of the IoT applications rely on IoT devices and services distributed in multiple IoT environments to satisfy users' demands and improve their functionalities. Data should only be sent by authorized users and be accessed by authorized users, during data sharing among multiple IoT environments.

The second challenge that we want to address in this thesis is to regulate data sharing within an IoT environment under ordinary and emergency situations. Recent emergencies, such as the COVID-19 pandemic, have shown that proper emergency management should provide data sharing during an emergency situation to monitor and possibly mitigate the effect of the emergency situation. IoT technologies provide valid support to the development of efficient data sharing and analysis services and thus appear well suited for building emergency management applications. In addition to this, IoT has magnified the possibility of acquiring data from different sensors and employing these data to detect and manage emergencies. However, IoT has also amplified the possibility of information misuse and unauthorized access to information by untrusted users. Thus, an emergency management application in an IoT environment should be complemented with a proper access control approach to control data sharing against unauthorized access.

In this dissertation, we do a step to address two open research challenges related to data protection in IoT environments which are briefly introduced above. To address these challenges, we propose two access control frameworks rely on Attribute Based Access Control(ABAC) model: the first one regulates data sharing among interconnected MQTT-based IoT environments, whereas the second one regulates data sharing within MQTT-based IoT environment during ordinary and emergency situations.

The key contribution of the first framework in the thesis is to manage the access to data generated and exchanged within interconnected IoT environments, which distinguishes the proposed framework from the majority of approaches in the literature which control a single IoT environment. The key contribution of the second framework in the thesis is to manage controlled and timely data sharing in both emergency and ordinary situations in an MQTT-based IoT environment. Both proposed access control frameworks target MQTT-based IoT environments since the MQTT protocol is widely adopted within IoT applications and used in various IoT scenarios. We also experimentally analyze the proposed frameworks to present their efficiencies. Early experimental performance evaluations show promising results and a quite acceptable policy enforcement overhead for each framework.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet of Things (IoT) refers to a network where physical smart entities referred to as things communicate with each other over the internet [8]. The integration of physical and smart things creates a massive ecosystem, whose scale could not be reached by traditional systems. More specifically this massive ecosystem contains seamless interconnection among billions of actuators, embedded systems, resources, and users.

IoT applications, which enable devices, companies, and users to join this ecosystem, are growing in popularity since they increase our lifestyle quality day by day [10]. For instance, by exploiting the pervasivity of wearable technologies, several IoT applications assist users during their daily routines (e.g., sport training or health monitoring).The report of IoT Analytics[1] forecasts the diffusion of IoT devices and applications to be increased and projects the IoT ecosystem to consist of more than 27 billion interconnected devices by 2025.

Though the benefits of IoT applications are manifold, due to the nature of the managed data, IoT applications have been recognized as potential threats to user privacy and data confidentiality (e.g., see [1]). Insufficient security protection mechanisms in IoT applications can cause unauthorized entities to access personal data[2]. Since current IoT environments allow both private companies and governments to make use of data on an unprecedented scale in order to carry on their activities, strong data protection frameworks should be devised. These frameworks should provide appropriate data security and confidentiality, and prevent data from being accessed by unauthorized entities.

In recent years, several researchers have deeply analyzed the trade-off between service utility and user privacy and confidentiality (e.g., [54]) by proposing several data protection solutions. In particular, in order to solve security issues related to unauthorized access, many access control systems have been proposed in both academic and industrial environ-

[1]https://iot-analytics.com/number-connected-iot-devices/

[2]According to General Data Protection Regulation(GDPR) [69], which is comprehensive legislation proposed by European Union, personal data is sensitive data that is represented as any information (such as ethnic origin, political opinions, religious beliefs, genetic data, biometric data, healthcare data) concerning identified or identifiable natural person (see Art. 4 GDPR Definitions. Online:https://gdpr-info.eu/art-4-gdpr/).

ments that ensure that only authorized subjects can access the protected resources. More specifically, the main purpose of access control systems is to determine who can access specific resources under which circumstances via the access control policies.In [73], an access control system is divided into three main components: a *policy*, which refers to the authorization requirements that should be satisfied to grant access privileges to subjects, a *model*, which refers to the formal presentation of the policy enforced by the access control system, whereas a *mechanism* refers to the implementation of the model. Therefore, an access control system involves the following steps: defining an access control *policy* (set of access control rules), selecting an access control *model* to encapsulate the defined *policy*, implementing the *model* via the proposed *mechanism*, and enforcing the set of access control rules to determine access control [63].

Access control models have been proposed also for IoT environments to protect resources from unauthorized entities (we survey the most relevant ones in Chapter 3). Unlike many other environments, the IoT scenario is pretty challenging in providing proper security and privacy solutions due to its main characteristics, that is, dynamicity, that refers to the fact that the network topology and connectivity of IoT devices can be constantly changing, and the massive amount of resources to be protected. Other characteristics of IoT ecosystems are also relevant as well for access control purposes, such as being heterogeneous, intelligent, resource constraint, scalable, and latency-sensitive [63, 68].

Even though traditional access control models (e.g., Discretionary Access Control Model) have been employed for other environments, these models encounter difficulties in addressing the dynamicity of IoT environments [48].

Therefore, additional solutions have been proposed for IoT applications, mainly based on the following families of access control models: Capability-Based Access Control(CapBAC) [42], Usage Control(UCON) [66], Role-Based Access Control(RBAC) [74], and Attribute-Based Access Control(ABAC) [48].

Among the solutions based on the above mentioned models, the proposals which are based on Attribute-Based Access Control (ABAC), have been widely adopted in the last years [67]. ABAC models provide dynamic, flexible, and context-aware access control. These characteristics perfectly fit the IoT environments.

Even though the current access control systems tailored for IoT environments in the literature handle data sharing among IoT devices by employing various access control models and mechanisms, two important challenges have still not been sufficiently examined: namely *regulating data sharing within interconnected IoT environments* and *regulating data sharing in emergency situations within an IoT environment.* Therefore, in this thesis, we mainly focus on addressing these challenges properly.

In this thesis, we leverage on ABAC to regulate the reception and the publishing of messages exchanged within IoT environments. Our proposal relies on an ABAC framework proposed in [22], which regulates data sharing within a single IoT environment.

Let us start by the need of regulating data sharing within interconnected IoT environments.

The majority of proposed access control frameworks in the literature aimed at regulating the access to data generated and exchanged within a single IoT environment by adopting centralized enforcement mechanisms. However, currently, most of the IoT applications rely on IoT devices and services distributed over multiple IoT environments to satisfy users' demands and improve their functionalities. This has resulted in research work proposing frameworks(e.g., [85, 89]) on support of distributed IoT scenarios, such as for instance the Internet of Vehicle (IoV) which consists of traffic lights, services, cars, and pedestrians.

Though the approaches based on distributed architectures allow the delivery of more advanced services to users than the approaches based on the centralized architecture, they also bring serious security/privacy threats, as they extend the scope of the sensed data to multiple environments. Hence, data should only be sent by authorized users and be accessed by authorized users, during data sharing among multiple IoT environments.

An access control framework, which regulates data sharing within multiple IoT environments should provide fine-grained access control for shared data among environments to satisfy the principle of least privilege (PoLP), which refers to minimum levels of access or permissions that are given to a user to perform his/her job functions. The fine-grained access control avoids more resources than those strictly requested could be accessed among multiple environments.

The access control framework should also allow users to have more control over their data, in the case the users wish to set additional restrictions on their data. More specifically, though the users' data can be forwarded to the other IoT environments by means of satisfaction of the access control policies, the access control framework should give users a chance to restrict privileges that are already granted by access control policies specified by security administrators.

Lastly, the access control framework should be context-aware.

These requirements and the challenges explained above motivate us to propose an ABAC framework [24] which regulates data sharing among interconnected MQTT-based IoT environments. To the best of our knowledge, regulating data sharing across interconnected IoT environments has not been analyzed sufficiently in the literature, only [37] have targeted this issue.

The access control framework proposed in this thesis targets MQTT-based IoT environments, since the MQTT protocol is widely adopted within IoT applications, used in various IoT scenarios. MQTT is a standard application layer protocol that enables the communication of IoT devices by means of the publish/subscribe architecture.[3] Due to the pervasivity of many IoT scenarios, the approach proposed in this thesis provides support for fine-grained, context-based access control policies. For instance, let us consider the case of an MQTT-based Internet of Sport (IoS) application. When deployed on exercise bikes, the application allows gym frequenters to participate in cycling races, and share data, such as the current speed, and the covered distance by means of MQTT messages. In such a scenario, it would be useful to specify an access control policy that regulates data sharing across gyms in order to limit the exchange of rider performances to the race duration, granting access only to the set of users registered for the race.

In addition, our access control framework gives users more control over their data by

[3]Online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

specifying their own preferences, in the case the users wish to set additional constraints on their data usage. User preferences restrict privileges already granted by access control policies. For instance, rider Mary wishing to enforce a stricter privacy protection, may specify a user preference that, during a competition, filters out the identifiable data from the data shared with the riders of other gyms.

Our access control framework extends the one proposed in [22], which introduces an ABAC framework to control the communication of devices operating in a single MQTT-based IoT environment. In the ABAC framework proposed in [22], access control policies are defined by the security administrator, whereas, user preferences are defined by end users, and restrict privileges granted by access control policies. Access control policies and user preferences are enforced through a centralized enforcement monitor that operates as a proxy between a server and multiple clients in a single MQTT-based IoT environment. The key contribution of the access control framework proposed in this thesis is to provide a decentralized approach to enforce access control policies and user preferences, which supersedes the centralized enforcement mechanism proposed in [22] (see Chapter 4 for more details).

Now let us briefly discuss our motivation behind addressing the other considered challenge, which is related to *regulate data sharing in emergency situations within an IoT environment.*

Recent emergencies, such as the COVID-19 pandemic, have shown that, due to scarcely information sharing, emergency protocols often fail in fully achieving their goals. For instance, during the COVID-19 pandemic, contact tracing has been pointed out by the World Health Organization as a strategic tool for contrasting SARS-CoV-2 diffusion and reducing COVID-19 mortality. However, manual contact tracing methods proved scarcely applicable, highly demanding in terms of time and human resources, and overall impractical with a high number of new daily cases.

Contact tracing apps have addressed the scalability and performance issues of manual methods. However, due to a scarce perception of the enforced data protection constraints, in several western countries, citizens proved unavailable to install and use these apps [2]. As a consequence, the efficacy of contact tracing has been undermined by limited population coverage. These facts suggest that efficient data sharing is a key requirement for emergency management.

Efficient emergency management starts with timely identification of an emergency through the analysis of what has occurred in a target scenario and requires that all resources needed to properly handle the identified emergency could be timely accessed by authorized subjects. IoT technologies provide valid support to the development of efficient data sharing and analysis services and thus appear well suited for building emergency management applications. In addition to this, IoT has magnified the possibility of acquiring data from different sensors and employing these data to detect and manage emergencies.

The management of an emergency in an IoT environment typically requires granting exceptional privileges to users, which in an ordinary situation would not be permitted. For instance, in an ordinary situation, a physician responsible to provide treatment has to ensure that valid consent has been obtained from the patient or a delegated person before the treatment can begin. However, if an emergency occurs and the treatment is finalized to save the patient life, it can be provided without consent. Nonetheless, all granted exceptional privileges have to be immediately revoked as soon as the emergency is over.

Though a variety of access control approaches for IoT applications have been proposed in the literature, just a few of them allow regulating data sharing in an emergency situation. Almost all of these proposals rely on a permission management approach known as *break the glass (BtG)* [18]. BtG enables users to request and then gain access to resources that would not be permitted to him/her in ordinary situations. Though BtG provides flexible emergency management, it has two main drawbacks. First, accesses executed after breaking the glass should be traced for later reviews to determine whether they caused possible information leakage [75]. Second, the abuse of BtG policies can lead the system to an unsafe state [21].

To avoid the weaknesses faced in the BtG approach, an alternative approach can be employed where data sharing during emergency situations within an IoT environment is regulated by the enforcement of proper emergency policies. Emergency policies can grant users all privileges needed for the management of specific emergencies as soon as they occur, instead of depending on the users' requests for exceptional access privileges. This approach favors a more efficient control of the protected data and does not put the system in an unsafe state. However, designing a model fitting this approach and designing a corresponding enforcement monitor is quite challenging due to the fact that the following requirements regarding emergency management and access control in an IoT environment must be satisfied:

- Since users are granted privileges as soon as the emergencies occur, the application which realizes this approach should detect emergencies through the analysis of what has occurred in an IoT environment.
- Two types of policies should be supported: ordinary and emergency access control policies. Emergency and ordinary ABAC policies are employed to regulate data sharing in emergency and ordinary situations respectively and emergency policies always supersede ordinary policies during emergency situations.
- The access control system should regulate data sharing with fine-grained access control during both ordinary and emergency situations.

These requirements and the challenges explained above motivate us to propose an ABAC framework to regulate data sharing within MQTT-based IoT applications in ordinary and emergency situations. To the best of our knowledge, none of the previous approaches in the literature propose an access control framework for IoT environments that both detect emergencies when they occur and support the enforcement of emergency policies.

Similar to the proposed framework which regulates data sharing within interconnected IoT environments, the proposed framework, which regulates data sharing in emergency situations within an IoT environment, employs the ABAC framework proposed in [22] as its core and extends the framework in [22] with various features. The key novel features introduced by the framework proposed in this thesis, include:

- modeling support required to: i) define the events that trigger an emergency, ii) bind events to MQTT messages, iii) specify emergency situations along with their possible evolution, and iv) specify emergency policies;
- emergency management functionalities, such as the ability to: i) detect occurrences of modeled events starting from the analysis of MQTT control packets exchanged in a monitored application, and ii) identify the possible evolution of emergency situations. For event detection, we leverage on a complex event processing (CEP) engine;
- access control capabilities, such as the ability to enforce both regular and emergency access control policies which apply to an access request issued in a specific context.

We will discuss the details of the proposed framework in Chapter 5.

1.1 Contributions

In summary, this dissertation provides the following main research contributions:

- An extension [24] of the core framework proposed in [22]. [24] provides a decentralized approach to regulate data sharing across interconnected MQTT-based IoT environments. The extended framework introduces: i) new access control policies and user preferences to regulate data sharing across interconnected MQTT-based IoT environments, which can be specified in any environment of the interconnected pair, ii) a new enforcement monitor which regulates the messages that can enter/leave an environment, iii) a decentralized enforcement mechanism, leveraging on the joint work of monitors deployed in each environment of a bridged pair, and along the bridge, iv) lastly, an experimental evaluation of the framework performance.
- Another extension of the core framework proposed in [22]. to enforce controlled data sharing within MQTT-based IoT ecosystems during both emergency and ordinary situations. The proposed framework analyzes the MQTT messages exchanged in a monitored ecosystem leveraging on Complex Event Processing for emergency detection. Emergency and ordinary ABAC policies are employed to regulate data sharing in emergency and ordinary situations respectively by means of a new enforcement mechanism. We also assess the feasibility of the proposed approach with a case study related to a healthcare application that monitors nursing home patients during the COVID-19 pandemic.

1.2 Thesis Organization

This dissertation consists of six chapters and two appendices organized as follows:

- Chapter 2 presents fundamental information about the MQTT protocol along with the access control framework proposed in [22]. The frameworks proposed in this thesis extend the access control framework proposed in [22].
- Chapter 3 presents the state-of-art about access control within IoT environments and discuss pros and cons of the main proposals.
- Chapter 4 presents our ABAC-based access control framework that regulates data sharing among interconnected MQTT-based IoT environments.
- Chapter 5 presents our ABAC-based access control framework that regulates data sharing within an MQTT-based IoT environment during ordinary and emergency situations.
- Chapter 6 concludes the thesis and discuss future research directions.
- Appendix A presents the abbreviations that are employed within this thesis.
- Appendix B briefly presents the publications which contain the results presented in the thesis.

1.3 Related Publications

The research activities explained in this dissertation have brought to the following publications:

- Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. ”Regulating data sharing across MQTT environments.” Journal of Network and Computer Applications 174 (2021): 102907.
- Efficient ABAC based information sharing within MQTT environments under emergencies
- Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. ”Access Control Enforcement in IoT: state of the art and open challenges in the Zero Trust era” in Proc. of the Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (IEEE TPS 2021), December 13 - 15, 2021.

Chapter 2

Background

In this chapter, we present some basic concepts related to the MQTT protocol [12] along with the access control framework proposed in [22]. Such framework in instrumental to help the reader to understand the proposed frameworks in Chapters 4 and 5. [22] provides an ABAC framework to control the communication of devices operating in a single MQTT-based IoT environment.

2.1 MQTT

MQTT is a standard application layer protocol that enables the communication of IoT devices by means of the publish/subscribe architecture. MQTT protocol is widely adopted within IoT applications and used in various IoT scenarios [59].

In an MQTT environment, MQTT clients communicate with other clients by means of a message broker and the communication can only be achieved if clients are connected with the MQTT broker. Once MQTT clients connect to the MQTT broker, they can either request to publish application messages through the MQTT broker on given topics or they can subscribe to the reception of messages on the topics matching a topic filter expression.

A topic is a string structured as a sequence of alphanumeric tokens, referred to as topic levels, and separated by a topic level separator ("/" is the default topic level separator in MQTT specification).

Example 2.1. *Let us consider the case of an Internet of Sport (IoS) MyPersonalTrainer, developed for MyGymBrand, a brand of affiliated sport halls, which allows the gym frequenters to share data about training sessions. In particular, when deployed on exercise bikes, the app allows them to participate to cycling races, sharing data such as the current speed, and the covered distance. Suppose that the smart treadmills of the sport halls, which integrate multiple sensors and a tablet, also host an instance of MyPersonalTrainer. Messages published by the app on the runner performances in a training session ts can refer to the following topics:*

- *tr/performance/ts/speed,*

- *tr/performance/ts/avgspeed,*
- *tr/performance/ts/length,*
- *tr/performance/ts/duration,*
- *tr/performance/ts/hearthbeats.*

where tr and ts are placeholders for a treadmill and a training session identifiers, respectively.

A topic filter is a textual expression that can be structured as a sequence of topic levels, where each of the levels specifies an alphanumeric token or the wildcard characters "+" or "#", which allow referring to multiple topics. "+" wildcard character is employed for single topic level matching, whereas, "#" wildcard character is employed to match any level in a topic.

Example 2.2. *Let us consider the Internet of Sport environment introduced in Example 2.1. Suppose that the devices introduced in Example 2.1 have also been configured to subscribe to the receiving of performance data of all users attending ts, specifying +/performance/ts/+ as a topic filter. In addition, the app deployed on gym coaches' tablets allows them to monitor the performance data of gym frequenters. The app has been configured to subscribe to the topic filter +/performance/#, so that coaches can receive performance data published by any treadmill within any training session.*

Once an MQTT broker receives a publishing request on a topic t, the broker forwards the message[1] to any client who has subscribed to a topic filter expression that matches with t.

The communications among an MQTT broker and clients are achieved by exchanging control packets [12]. The control packets defined by MQTT are shown in Table 2.1.

Let us briefly explain how the communications among an MQTT broker and clients are regulated by the MQTT protocol. Suppose an MQTT client c sends a connection request cp_{CN} to the MQTT broker b with the aim of sending or receiving messages. Once this request is sent to b, b evaluates the request and sends back an acknowledgement cp_{CA} to c which specifies whether the request has been accepted. In the case the request has been accepted, the connection among b and c is opened. After the connection is established, c can request to publish an application message on a topic t with a payload p by issuing a publishing request packet cp_{PB}, or c can request the reception of messages on topics that match a topic filter tf, by sending a subscription request packet cp_{SB} to the broker. The broker b should send back to c the related acknowledge (e.g., cp_{PA}, cp_{SA} etc...) depending on the request type.

MQTT protocol employs three levels of Quality of Service (QoS) to guarantee the reliability of message delivery: *most once (0)*, *at least once (1)*, or *exactly once (2)*. Clients

[1]For the messages, a predominant data-interchange format adopted in numerous MQTT based applications is JSON [17].

Table 2.1: MQTT control packets

Control Packet	*Acronym*	*Description*
CONNECT	cp_{CN}	Connection request
CONNACK	cp_{CA}	Connect acknowledgment
PUBLISH	cp_{PB}	Publish message
PUBACK	cp_{PA}	Publish acknowledgement
PUBREC	cp_{PRC}	Publish received
PUBREL	cp_{PRL}	Publish release
PUBCOMP	cp_{PC}	Publish complete
SUBSCRIBE	cp_{SB}	Subscribe to topics
SUBACK	cp_{SA}	Subscribe acknowledgement
UNSUBSCRIBE	cp_{US}	Unsubscribe from topics
UNSUBACK	cp_{UA}	Unsubscribe acknowledgement
PINGREQ	cp_{PRQ}	PING request
PINGRESP	cp_{PRS}	PING response
DISCONNECT	cp_{DS}	Disconnect notification

can specify their messages with QoS-0 level if missing or undelivered messages are tolerated, otherwise. In contrast, QoS-1 level or QoS-2 level can be employed to guarantee message retransmission and delivery by means of dedicated control packets. An interested reader can refer to the MQTT specification [12] for further details on the MQTT protocol.

As we mentioned in Chapter 1, the first challenge that we want to address in this thesis is to *regulate data sharing within interconnected IoT environments.* MQTT-based IoT environments, which are configured with the brokers, can be connected together by means of one of the advanced features of MQTT protocol, that is, *bridging*, which allows data sharing among interconnected environments. Basically, the bridging mechanism enables to connect two or more brokers together and for bridging, one of the brokers is assigned as a *bridging broker*, that is configured to connect to a target remote broker, with which it shares messages published by clients of the respective environments. The interaction is regulated by means of *bridging rules.*

A bridging rule *br* specified for a bridging broker *bb* is a tuple *<rb, tp, dr, qos, lp, rp>*, where *rb* denotes the remote broker representing the target of *bb*, *tp* is a topic filter expression, specifying the topics of the messages to be shared by *bb*; *dr* specifies the sharing direction (*in*, *out*), which is *in*, if *br* allows *bb* to receive messages from *rb*, or *out*, if *br* allows *bb* to send messages to *rb*; *qos* specifies the QoS level to be used for the message exchange; finally, *lp* and *rp* are local and remote prefixes, which are employed by *bb* to remap the topic of outgoing and incoming messages, respectively. If *br* specifies in as sharing direction, *bb* prepends *tp* with *rp* and subscribes to the resulting topic on *rb*. On receipt of a message *m* from *rb* whose topic matches the subscribed topic, *bb* substitutes the remote prefix *rp* prepending the topic of *m* with *lp*, and forwards *m* to the local clients who subscribed to a matching pattern. In contrast, if *br* specifies out as sharing direction, *bb* prepends *tp* with *lp* and subscribes the resulting topic on the local broker. On receipt of a message *m* published by a local client, whose topic matches the subscribed topic, *bb* substitutes the local prefix with *rp*, and forwards *m* to *b*.

Example 2.3. *Suppose that MyPersonalTrainer enables users to share performance data with a remote analysis server, that allows comparing them with previously tracked data of the same user, of users attending the same course, of frequenters of the same gym, or runners attending any sport hall. Suppose that MyGym (can be considered as a local environment) hosts a broker, which handles the communication among its gym apparatus, as well as with brokers of the associated gym RemoteGym (that can be considered as a remote environment), and of the data analyzer environment, which hosts the remote analysis server. Examples of bridging rules are the following:*

- *br1 = <Analyzer, +/performance/+/speed, out, 0, "", MyGym>.*
- *br2 = <RemoteGym, +/performance/+/speed, in, 0, "", "">.*

br1 enables the forwarding to the analyzer environment of messages specifying the current speed of a runner that attends a training session. The remote prefix MyGym denotes the provenance of the message. The remote analysis server is a client of the analyzer environment which subscribes the receiving of messages on topics that match the filter +/+/performance/#, which allows the server to access performance data of the frequenters of any gym. In contrast, on the basis of br2, MyGym broker subscribes the receiving of messages from RemoteGym that refer to the speed of a runner attending any training session of the associated sport hall. The remote analysis server is a client of the analyzer environment which subscribes the receiving of messages on topics that match the filter +/+/performance/#, which allows the server to access performance data of the frequenters of any gym.

2.2 ABAC framework within a single MQTT-based IoT environment

In this section, we briefly explain the access control framework proposed in [22] that provides the base structure for our proposed frameworks. The access control framework in [22] aims at regulating MQTT clients' communications in a single IoT environment. The solution in [22] relies on an ABAC model for MQTT environments. ABAC has been used in [22] since ABAC provides outstanding flexibility, as well as for the dynamic and context-aware nature of the supported policies, these characteristics perfectly fit for the IoT environments. The framework consists of an enforcement mechanism designed for the model, and implemented by an enforcement monitor, which can be easily integrated into MQTT-based environments. In [22], ABAC is employed to regulate the reception and the publishing of MQTT messages, on the basis of *access control policies* and *user preferences.*

2.2.1 The ABAC Model

The ABAC model proposed in [22] relies on the following three main concepts:

- a *subject* who sends access requests,

- a *protection object*, and
- the *context* within which an access request has been issued.

A subject s represents an MQTT client, which, possibly on behalf of a user, connects to an MQTT broker in order to send or receive messages. s is characterized by attributes, such as client identifier and optionally user identifier, which represent user on behalf of whom the client operates. Subjects who have similarities can be classified into subject groups.

Application messages are the protection objects of the considered model and they are characterized by attributes that model message properties, which can be employed for access control purposes. More specifically, an object o is comprised of an attribute t, which specifies the topic of the message modeled by o, and another attribute *pl*, which specifies the message payload.

Lastly, the environment e represents the context within which an access request is issued, and could be characterized by attributes such as location, time, and access purpose.

The subject, object, and environment are denoted *attribute types* in the ABAC model proposed in [22] and an attribute related to one of these attribute types is represented with the composition of the following three elements:

- the reference of its attribute type (the references of the subject, object, and environment attribute types are represented with s, o, and e respectively),
- a separator (the dot sign is assigned as the separator),
- and the name of the attribute.

For instance, *role* attribute can be defined as one of the attributes related to the subject attribute type and, thus, can be notated as *s.role*.

Data sharing is regulated on the basis of ABAC policies, specified by security administrators, which grant subjects the read/write access to messages on the specific topic(s). Read and write accesses respectively denote the privileges to send/receive messages on given topics.

Access control policies grant privileges under satisfaction of boolean expressions, denoted as *parametric predicates*, built by composition of subject, object and environment attributes, and sets of predefined operators and functions[2].

Definition 2.1 (Access control policy [22])**.** *An access control policy p is a tuple ⟨s, tf, exp, pr⟩, where s refers to the subjects constrained by p, tf specifies a topic filter expression, exp is a parametric predicate, whereas pr specifies the read /write privileges granted to s if exp is satisfied.*

[2]We consider mathematical operators ($>$, $<$, $=$, $+$, -, *, /,%), logical operators ($\wedge$, $\vee$, $\neg$), set operators ($\in, \subset, \subseteq, \cap, \cup, \setminus$), logical quantifiers ($\forall$, $\exists$), and predefined functions that allow the processing of attributes values.

Example 2.4. *Let us consider a policy p_1 which grants frequenters enrolled to a gym course the privilege to publish performance data and read performance data of other trainees, and a policy p_2, which allows gym coaches to access performance data of gym frequenters during their working hours. p_1 can be specified as: ⟨frequenter, +/performance/ts/+, isEnrolled(s.sid),rw⟩, where isEnrolled(s.sid) is a function that checks whether the subject represented by the sid identifier is enrolled in a gym course, whereas p_2 is modeled as ⟨gymcoach, +/performance/+, isWorkingTime(t,s.sid), r⟩, where isWorkingTime(t,s.sid) is a function that checks whether the time referred to by t matches the work shift of the subject.*

In addition to access control policies, the model proposed in [22] supports user-defined policies that allow a user to further constrain the read privileges granted by the access control policies. This type of policies is named as*user preferences.* User preferences constrain the access to messages published by a user, on the basis of parametric predicates.

Definition 2.2 (User preference [22])**.** *A user preference up is a tuple ⟨uid, tf, sub_exp⟩, where uid specifies the identifier of a user who wishes to protect the access to messages published by any of the clients he/she handles, tf specifies a topic filter expression which refers to the topics of the messages published on behalf of uid to which up applies, whereas sub_exp is a parametric predicate specifying a precondition to the receiving of these messages.*

Example 2.5. *Assume that Mary, who is a frequenter of MyGym, limits to Alice, who is a coach of MyGymBrand, the privilege to access Mary's performance data related to the training session ts by specifying user preference up_1. The user preference up_1 can be defined as up_1=⟨Mary, +/performance/ts/+,s.rid="coach"∧ s.uid="Alice"⟩.*

2.2.2 Access Control Enforcement

Now let us briefly introduce the access control enforcement mechanism proposed in [22]. A high-level view of the system architecture proposed in [22] is shown in Fig. 2.1.

Access control policies and user preferences are enforced through an enforcement monitor that operates as a proxy between an MQTT message broker and multiple clients, and a key–value datastore, which manages access control policies and user preferences to be enforced by the monitor. The enforcement monitor is the only component that can connect to the server, on behalf of the client.

A publishing request, issued by a client, is intercepted by the enforcement monitor and, if at least one applicable policy is satisfied, the enforcement monitor grants the requested privilege to the client, otherwise, the publishing request is denied and the application message is blocked. If the publishing operation complies with at least one policy, the monitor derives the preferences specified by the user on behalf of whom the publishing has been requested. Then the monitor embeds the derived preferences into the payload of the publishing request and forwards the packet to the broker.

Publishing requests issued by the broker are handled in a similar way. The monitor extracts attributes and user preferences from the control packet's payload, derives the subject,

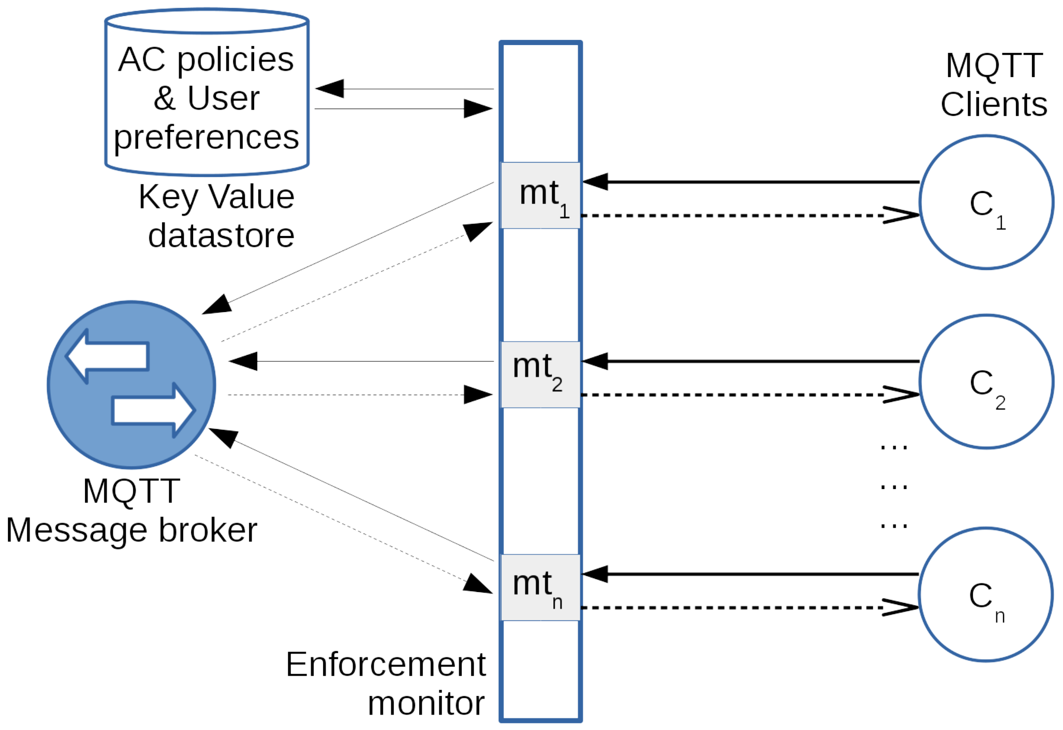

Figure 2.1: A high-level view of the system architecture in [22]

object, and environment attributes modeling the context within which the message should be received by the subscriber, and evaluates the preferences wrt the derived attributes. If no preference is satisfied, the request is blocked, whereas if at least one preference is satisfied the monitor derives the applicable access control policies. In case the read access complies at least with one of the policies, the monitor removes from the payload all previously embedded metadata and forwards the packet to the subscriber, otherwise, it blocks the request.

Example 2.6. *Let us again consider the case of an Internet of Sport (IoS) introduced throughout this chapter. Suppose that the MyPersonalTrainer app, which is hosted by a smart treadmill of a sports hall, issues a publishing request (cp_{PB}) related to the gym frequenter Mary's average speed in a training session (ts) after the initial connection of the app is established. Assume that gym frequenter Mary is already enrolled in a gym course and Alice, who is a coach of Mary, wants to monitor Mary's performance along with other frequenters' performances during her working time. Moreover Mary limits to Alice the privilege to access Mary's performance data related to the training session ts by specifying user preference up_1 which is introduced in Example 2.5. Assume that topic t of cp_{PB} is "tr/performance/ts/avgspeed" and payload pl of cp_{PB} is {"s.sid":Mary, "avgspeed":7.50}* *which is represented as a JSON object*[3]*.*

An enforcement monitor (em) intercepts cp_{PB} *and derives the access control policies that should be applied to regulate the processing of this request. Assume that a policy* p_1*, which is introduced in Example 2.4 is selected by em to authorize the publishing of* cp_{PB}*. In addition to this, em derives the* up_1 *specified by Mary and then embeds the derived preference into the pl of* cp_{PB} *(let us call new payload* pl^x *and new control packet* $cp_{PB}x$*), and forwards* $cp_{PB}x$ *to the broker.*

Publishing requests issued by the broker are handled in a similar way. Assume that coach Alice is already connected to the system and subscribes "+/performance/+" to monitor performances of all gym frequenters(including Mary). em extracts attributes and user preferences from the pl^x*, derives the subject, object and environment attributes modeling the context within which the message should be received by Alice, and evaluates* up_1 *wrt the derived attributes. Since* up_1 *is satisfied, em derives the applicable access control policies. Assume that a policy* p_2*, which is introduced in Example 2.4 is selected by em to authorize the receiving of* cp_{PB}*. Before Alice receives the message,em removes all previously embedded metadata from the payload (it becomes pl again) and forwards* cp_{PB} *to Alice.*

[3]As we mentioned in Section 2.1, a predominant data-interchange format adopted in numerous MQTT based applications is JSON [17].

Chapter 3

Related Work

In this chapter, we review the state-of-the-art papers in the literature that deal with access control for IoT environments. The review analysis that has been conducted in this thesis relies on multiple aspects of access control solutions. More precisely, we consider three key aspects to classify the characteristics of the papers we surveyed:

- *Access control model* refers to the adopted authorization model employed in the paper. The traditional access control models, which rely on access control lists or access control matrices [53], are not considered in this chapter due to the fact that these models have been criticized in terms of the lack of expressiveness and dynamicity, which are needed in IoT environments [48]. Thus, we only consider the following popular access control models: Capability Based Access Control(CapBAC) [42], Usage Control(UCON) [66], Role Based Access Control(RBAC) [74] and Attribute Based Access Control(ABAC) [48] models, and enforcement solutions (e.g., Attribute Based Encryption(ABE) [40]) based on these models.

  CapBAC models employ capability tokens that give the possessor permissions to access target resources. More specifically, authorizations to access specific objects are materialized into capability tokens, which are assigned to rightful subjects. In order to access a target resource, a subject has to prove his/her authorization by presenting the related capability token. In the CapBAC model, the entity that wants to access a target resource is denoted *grantee* and the resource owner entity that can issue the capability tokens to the grantees is denoted *granter*. A capability token typically consists of the granted rights, the target resource, the identity of the grantee, and necessary information for the access such as capability validity period. After a capability token issued by the granter send to a grantee, the grantee can access the target resource owned by the granted. Before access privilege is granted to the grantee, the capability validity period of the capability token, and credentials of the grantee should be checked whether they are valid, and if there is no issue, the access privilege is granted.

  UCON model is typically comprised of six elements: subjects & subject attributes, objects & object attributes, access rights, authorization rules, obligations, and conditions. In the UCON model, both subject and object attributes can be mutable attributes: subject attributes can be changed as a consequence of subjects' actions (e.g., user credit balance) and object attributes can be changed during access (e.g., last access time). An access right represents access privilege that a subject can hold on an object. Authorization rules, obligations and conditions are functional predicates that are employed for access control decisions. More specifically, authorization rules have to be evaluated for access control decisions based on subject and object attributes and the requested specific right. Obligations are employed to verify mandatory requirements a subject has to perform before or during an access control decision. Lastly, conditions are environmental attributes that are independent from the attributes of the subjects and objects. The key features of the UCON model are the mutability of attributes and continuity of enforcement. The first one represents that attribute values can be modified as side-effects of subjects' actions or updates of the object's properties, whereas, the second one represents that continuous policy evaluation that is made in pre-access, during-access, and even post-access phases.

RBAC is an access control model for controlling user access to resources based on roles. Roles are created according to the job function and users are given roles according to their responsibilities. In the RBAC model, roles are often organized in a role hierarchy, which defines the inheritance of permissions between roles. Users are not assigned permissions directly, but only acquire the permissions through their role(s), thus, permissions can be given to users by means of the success of the conditions of the policies that are used to assess the roles. The pure RBAC model, however, is criticized in [63] as being insufficient to model security policies that interpret complex IoT scenarios due to the well-known role explosion problem, thus, many RBAC solutions, which target IoT environments, extend the pure RBAC model.

ABAC, unlike many traditional models of access control, which rely on the manual assignment of roles, ownership, or security labels by a system, allows for the creation of policies based on the existing attributes of the users, objects and environments [76]. In the ABAC model, policies and access requests are specified in terms of attribute names/values pairs and authorizations are constrained to the satisfaction of conditions which refer to attributes of subjects, objects, and environment. The subject is granted appropriate access permissions by the system according to his/her attributes along with attributes of object and environment, when he/she issues an access request.

- *Architecture* refers to the access control architecture proposed in the paper. This can be either centralized or distributed. A centralized architecture relies on a single policy decision component that makes all decisions about access requests, whereas a distributed architecture relies on distributed mechanisms to manage and enforce the policies. More specifically, in a centralized architecture, the physical node sends the access request to a specialized policy decision point that provides centralized authorization. The policy decision point decides to whether grant or deny the privileges based on the policy enforcement. Contrary to a centralized architecture, the policy decisions are performed by several policy decision points deployed in different environments in a distributed architecture. In a distributed architecture, the policy decision components either are embedded into IoT devices or IoT devices are connected to the policy decision components that are independent of the devices.

Let us briefly discuss on advantages and disadvantages of both architectures. Generally, a centralized architecture for the IoT environments provides a high computational capability along with a strong memory for the authorization processes which eases the resource-constrained IoT devices from the burden of handling a vast amount of access control-related information. Another advantage of the centralized architecture for the IoT environments is to possibly offer easy administration for the system. However, since the centralized architecture relies on a single policy decision component, a compromised policy decision component may lead to a single point of failure. Moreover, loads of access requests can cause the system bottleneck and drop the general performance of the system [32].

A distributed architecture for the IoT environments possibly solves an issue related to the single point of failure for the policy decision components, more precisely, the distributed architecture may allow IoT devices to connect another policy decision component, once the existing one compromises. Another advantage of the distributed architecture is to possibly provide better contextual awareness in authorization than a centralized one since the policy decision components and devices are closer to each other more than in the centralized architecture. The most obvious drawback of the distributed architecture is to implement a complex authorization mechanism on the side of the resource-constrained devices, since these devices are not sufficient to handle access control logic. Another disadvantage of the distributed architecture is to handle local access control policies remotely.

- *Emergency Management* refers to the type of emergency management support provided by the paper. In this respect, we can distinguish among emergency management support based on emergency policies, emergency management support based on Break-the-Glass(BtG) policies or no emergency management support.

  Emergency management support based on BtG policies relies on a strategy that allows a user to request the resources during an emergency and then gains access to the resources which would not be permitted to him/her in normal conditions. More precisely, once a user sends break-the-glass access requests to the system during an emergency, the access decision is derived from the applicable BtG policies which bypass the policies that are employed during an ordinary situation. Once the emergency ends, the BtG accesses shouldn't grant additional privileges to the users.

  Though BtG provides flexible emergency management, it has two main drawbacks. First, the accesses executed after breaking the glass should be traced for later reviews to determine whether they caused possible information leakage [75]. Second, the abuse of BtG policies can lead the system to an unsafe state [21]. More precisely, if the ordinary access control policies aren't defined sufficiently to minimize the necessity of breaking the glass accesses, many users get more privileges than they should by employing BtG requests.

Emergency management support based on emergency policies: under this mode, for each of the considered emergencies, an emergency policy is specified which grants extra privilege when the corresponding emergency happens. Since emergency management plans are expected to be a priori defined, emergency policies could be specified in such a way to fulfill information sharing requirements elicited from the associated plan. Permission management based on emergency policies allows shorting data access time, since no request to override a permission has to be issued, and data can thus be received by authorized subjects as soon as the emergency begins. The approaches, which support emergency management based on emergency policies, should detect emergencies once they occur, and grant exceptional privileges to the authorized subjects during emergencies. Once the emergencies are over, only ordinary policies are involved in the access control decisions.

In what follows, we first introduce "classical" access control solutions that do not provide any emergency management support, then we analyze access control solutions tailored to the management of access control during emergency situations. The summary of the analysis that has been conducted is given in Table 3.1. In Table 3.1, *Reference* column shows to the references of the state-of-the-art access control solutions, whereas *Access Control Model*, *Architecture* and *Emergency Management* columns refer to the employed access control model, architecture and emergency management type for each state-of-the-art access control solution, respectively.

3.1 Classical Access Control Solutions

3.1.1 CapBAC-based Solutions

As we mentioned in the early part of this chapter, in the CapBAC model, authorizations to access specific objects are materialized into capability tokens, which are assigned to rightful subjects.

The CapBAC approach proposed in [42] employs a centralized enforcement mechanism for a single IoT environment. IoT device owners can grant to other subjects the access to the administered resources by assigning them properly defined capability tokens. Similarly, they can delegate all or part of the owned privileges to other subjects. Subjects can also revoke previously granted authorizations, by means of revocation tokens which specify the revocation time and reasons.

A distributed CapBAC framework was proposed in [46] (denoted as DCapBAC). In [46], IoT devices carry out the authorization processes by themselves, without the need for a central entity. To realize a distributed architecture, their framework allows IoT devices to make peer-to-peer interactions using CoAP [78]. More precisely, the access requests and the responses against these requests conveyed among devices are structured in the CoAP message format. In their framework, a capability token holds access control conditions assigned by its granter. Whilst a granter prepares the capability token, the granter signs its capability token by employing the Elliptic Curve Digital Signature Algorithm (ECDSA) [50] and the granter sends it to a grantee directly. Once the grantee obtains the capability token, it sends an access request with the capability token attachment directly to the granter. During the authorization process, the IoT device to which the capability token is sent authenticates the grantee and checks the validity of the token, the related ECDSA signature and the satisfaction of the access control conditions embedded in the capability token.

Bernabe et al. [14] have proposed a trust-based extension of DCapBAC, where authorizations are granted on the basis of the trust level associated with IoT devices. Device trustworthiness is computed on the basis of properties such as: i) the overall quality of services provided by the IoT devices (e.g.,throughput, delay), ii) devices reputation, and iii) possible relationships among the devices (e.g., group membership).

Hussein et al. [49] propose COCapBAC, a distributed CapBAC approach where authorizations are handled by IoT device communities, that is, IoT devices that share a common goal. The IoT devices, which are not resource-constrained, referred to as gatekeepers, generate an IoT device community and manage authorization decisions on behalf of resource-constrained devices in the same community. A subject sends an access request to the gatekeeper of the community and the gatekeeper forwards this request to a policy decision component. The policy decision component evaluates the request, generates a capability token, which contains access rights, and sends the capability token to the subject via the gatekeeper. Once the subject receives the capability token, he/she presents it to a gatekeeper, and the gatekeeper validates the correctness of the capability token against forgery from third parties.

Some works propose blockchain-based distributed CapBAC implementations (e.g., [56, 60, 86]).

For instance, BlendCAC [86] is a decentralized, context-aware CapBAC approach for IoT ecosystems, where smart contracts are employed to manage authorizations, IoT device registration, and access right revocation.In their framework, a subject can delegate his/her permissions to another subject by specifying the *delegation relationship*, which shows the permission delegation hierarchy among the subjects.An ordered list of delegation relationships for same object is called *delegation path*. All *delegation paths* for the same object are combined to construct a *delegation tree*, which will be employed during delegation operations. In the proposed delegation tree, a subject cannot obtain permissions from more than one subject. Once a subject registers himself/herself by adding his/her profile into the blockchain system, the subject can send the access request related to the target object to one of the dedicated policy decision components deployed in the blockchain environment. The policy decision component assesses the access request, and if the access request is accepted, the policy decision component sends the capability token by encoding the access right to the subject. After receiving the capability token from the blockchain system, the subject shows his/her capability token to access the target object.

Nakamura et al. [60] propose a CapBAC approach where Ethereum[1] [20] smart contracts are employed to manage and check the validity of capability tokens. Delegation relationships among subjects are traced in a delegation graph for each object, instead of a delegation tree proposed in [86]. The reason behind this structure change is to have more flexible capability delegation than a delegation tree usage in [86]. More precisely, in the delegation tree proposed in [86], the children nodes of the same parent node cannot delegate each other. Because in this case, the delegatee node needs to have two parents, however, each node is structured with a single parent [86]. On contrary, the delegation graph is capable of assigning more than one parent for each node, as a result, the permissions to a subject from more than one subject are possible.

Another blockchain-based CapBAC solution has been proposed in [56], where decentralized identifiers (DIDs) [2] are employed to manage the identities of IoT devices. The authors present the architecture of their framework and present a protocol that supports the authorization process, discussing the basic interactions between granter, grantee, and the on-chain smart contracts. In their framework, before authorization, both granters and grantees are required to register DIDs to identify themselves. Their framework supports two types of tokens: the first type is a typical *capability token* and the second type is a *device owner token* which shows that the ownership of granter to a specific device. DID registration, capability token and ownership token management are handled by smart contracts in their framework.

3.1.2 UCON-based Solutions

As we mentioned in the early part of this chapter, the mutability of attributes and continuity of enforcement are the key features of the UCON model.

La Marra et al. [52] proposed UCIoT, a distributed UCON framework for smart home IoT scenarios. Policy enforcement is carried out by IoT devices, leveraging on local attributes and remote attributes of other devices. In their framework, each node represents a smart IoT device logically connected to the others through a Distributed Hash Table. Their access control framework is embedded into each device in the smart home environment, and the devices can decide to permit or deny the accesses related to their resources. La Marra et al. [84] have also proposed a UCON based approach for MQTT-based IoT environments, which aims at addressing the weak authorization mechanism of these systems. In particular, once a subscription request of an MQTT client is authorized, such client can receive messages until it explicitly asks for cancelling the subscription. To address this issue, La Marra et al. [84] propose an MQTT broker which embeds an enforcement monitor supporting continuous policy evaluation. Supported policies refer to properties such as subscriber reputation, data reliability, environmental conditions, and allow a fine grained regulation of access privileges.

Dimitrakos et al. [29] have proposed UCON+, an extended distributed Usage Control

[1]https://ethereum.org/en/developers/docs/

[2]Decentralized Identifiers (DIDs) v1.0, W3C Proposed Recommendation, 03 August 2021 https://www.w3.org/TR/2021/PR-did-core-20210803/

model for IoT environments, which provides a trust-aware continuous authorization process based on UCON policies that also embed trust-based conditions. Let us assume that a policy can grant read privilege to a user about a resource if the user has a minimum trust level of 8 out of 10 before access is granted. Let's consider the same policy again, in addition to the above specification, the same policy is defined as follows: If the user can retain the minimum trust level at 7 out of 10 during resource usage, he/she can keep this privilege. In UCON+ subjects, object and context attributes are continuously monitored along with trust levels. Whenever a change of such mutable properties is detected, UCON policies are re-evaluated. The authors claim a low computational and memory footprint of their approach which makes it suitable to consumer IoT scenarios.

3.1.3 RBAC-based Solutions

As we mentioned in the early part of this chapter, in the RBAC model, the users are members of roles and the permissions are associated with roles.

Gwak et al. [43] have proposed a dynamic trust-based extension of RBAC for IoT. Authorizations are granted on the basis of roles, and the trustworthiness of the group of users to whom the same role has been assigned. Trust values of user groups are continuously derived from group members' behaviors. Granted access rights are revoked, if the trust value lowers below a minimum level.

Fernandez et al. [33] have complemented the authorization mechanisms of RBAC with OAuth 2.0 [45] based authentication. OAuth tokens are used to identify a subject who has sent an access request via an IoT device, along with his/her roles.

Bandara et al. [11] have proposed an RBAC-based approach for IoT smart building scenarios. A distinguishing feature of [11] is the provided support to multiple authentication techniques. Any authentication method shows a measure of how confident the system is that a just authenticated subject is who he/she claims to be [3]. Assume that, a user can authenticate himself/herself by employing one of the two authentication methods as follows: fingerprints and user-generated passwords. The fingerprints are more reliable proofs than passwords since the passwords can be stolen, thus, the confidence level of the fingerprint-based authentication method is higher than the password-based one. Resources are labeled with the minimum confidence level required for their accesses. Their enforcement mechanism evaluates the access request issued by authenticated subject, and grant the privilege to the subject if the confidence level of the authenticated subject is greater or equal to the resource's one and the role of the subject is the same as the required one.

Ameer et al. [6] have proposed EGRBAC, an extended RBAC model for smart home IoT environments. EGRBAC enhances the RBAC model with new conceptual elements referred to as device roles and environment roles. Device roles represent a group of IoT devices, whereas environment roles model environment states. A device role categorizes permissions of different IoT devices, while an environment role specifies environmental contexts. For instance, assume that in an smart home IoT environment, a smart oven might be dangerous for kids if they are alone at home. The device role of the smart oven can be defined as *dangerous devices.* Dangerous devices are always forbidden for kids, on contrary, the parents can use dangerous devices. Thus, the environment roles, which are employed in a policy to grant access privileges just to the parents for the dangerous devices, can be defined as *no-time* for kids and *all-time* for parents.

All RBAC-based access control frameworks, which are examined in this thesis, rely on centralized architecture.

3.1.4 ABAC-based Solutions

As we mentioned in the early part of this chapter, in the ABAC model, authorizations are constrained to the satisfaction of conditions which refer to attributes of subjects, objects, and environment.

Colombo and Ferrari [22] have proposed a centralized ABAC approach to regulate IoT device communication in MQTT environments on the basis of access control policies and user preferences. As a remainder, this ABAC framework provides the base structure for our proposed frameworks. In their framework, access control policies are defined by the security administrator, whereas, the user preferences are defined by the users, and the user preferences restrict privileges granted by access control policies. Access control policies and user preferences are enforced through an enforcement monitor that operates as a proxy between a server and multiple clients, and a key–value datastore, which manages access control policies and user preferences to be enforced by the monitor. The enforcement monitor is the only component that can connect to the server, on behalf of the client. Another centralized ABAC approach for MQTT environments has been presented in [38], but except for implementation strategies, the approach is aligned with the one originally presented in [22].

Dong et al. [30] propose a centralized state-aware ABAC framework for IoT environments. Their framework enriches the generic ABAC approach with the concept of *states* that describe the continuous actions of subjects to the resources. The authors give an example of why the use of state is needed in the smart IoT classroom environment. In the example, the authors consider that when the class is over, and no one stays in the classroom, all the former operations to the pieces of equipment in the room during the class should be revoked. The authors claim that most of the existing ABAC frameworks in the literature cannot deal with that revocation, since they do not hold previous states. In their framework, the states of continuous actions are changed depending on the actions performed by the users and they are recorded by the framework to control how users employ the resources after the users are authorized.

Carranza and Fong [37] propose a distributed ABAC framework for IoT environments that regulates data sharing among interconnected brokers. Their approach is built on top of an event-based architecture ([36]), which enables the communication of interconnected message brokers. Brokers interaction is regulated by scoping rules that constrain the range of the authorized receivers of any message to be routed. The approach relies on *brokering policies*, namely access control rules that constrain the brokers ability to propagate messages received through a channel to other channels. Their approach has been implemented exploiting an ad-hoc modified version of Mosquitto [3].

Some recent works propose ABAC implementations based on blockchain technology (e.g., [31, 44, 55]). The ABAC framework for IoT environments proposed by Liu et al. [55] is based on Hyperledger Fabric [7].[4] Each IoT device generates its resource URL and sends it to the blockchain. A user, who aims to access the resource, makes a request to the blockchain by proving his/her attributes. Three types of smart contracts are defined to support: i) the authorization process, ii) the management of ABAC policies, and iii) the management of device resources. The communication of IoT devices with the blockchain is mediated by smart gateways.

Hyperledger Fabric has also been employed in the ABAC framework proposed in [44]. Four types of smart contracts are used to: i) manage ABAC policies, ii) validate access requests and their responses, iii) manage private data in IoT, and iv) manage access records. IoT data are split into public and private data. Public data are stored in the cloud, whereas confidential and sensitive data are stored in the blockchain. After the grantee sends a request to access private data, a smart contract tries to authenticate the grantee by employing an access control policy defined by the granter. If this process is successful, the requested private data is obtained by the grantee. After the access, an access record is created based on the grantee's attributes and stored in a blockchain network.

Dramé-Maigné et al. [31] proposed a distributed ABAC framework based on blockchain technology and the trustworthiness of subject attributes. The authors introduce distributed components, namely Attribute Issuing Entities (AIEs) that are capable to access blockchain to retrieve requester subject' attributes and endorse them. Security administrators establish the trust values for AIEs in the initialization phase. Trust values of AIEs are employed to calculate the trust values of the subject attributes (e.g., if a trusted AIE endorses a subject attribute, the trust value of the subject attribute has a high trust value). A policy contains a minimum trust level value as a threshold assigned by the security administrator. The threshold of the policy is required to be passed by the computed trust values of the subject attributes of the requester and if all required attributes pass this threshold, access is granted. Their framework employs two types of smart contracts, where the first type enables the security administrator to update trust values of AIEs remotely, whereas the second type keeps track of users' attributes on the blockchain.

3.2 Access Control Solutions With Emergency Management Support

In this section we analyze state-of-the-art access control solutions that support emergency management based on either Break-The-Glass policies or emergency policies. The great majority of access control solutions that support emergency management do not explicitly target IoT environments, only a few exceptional works such as [87], [82] target IoT environments. Even though many access control solutions in the literature do not specifically

[3] https://mosquitto.org/

[4] https://hyperledger-fabric.readthedocs.io/en/latest/index.html

aim at functioning within IoT environments, they can be adapted to be used within IoT environments with minor updates. Thus, we enlarge our scope just for this section by not only considering access control solutions that target IoT environments but also considering access control solutions that do not explicitly target IoT environments. Note that, since Table 3.1 present the summaries of the state-of-the-art papers that only target IoT environments, we only insert in this table the access control solutions for emergency management targeting IoT environments.

3.2.1 Emergency Management Based on Break-The-Glass Policies

The great majority of approaches to handle access control during emergencies employ the break the glass (BtG) paradigm, according to which, during an emergency a user requests and gains access to resources which in normal situations would not be permitted.

A seminal work is the one by Brucker and Petritsch [18] that proposed an approach to integrate BtG policies into access control models. The proposed mechanism relies on BtG policies that extend the privileges granted by regular policies, allowing a fine grained control over protected resources. In case of break the glass requests during an emergency, the access decision is derived from the applicable active BtG policies. The same authors in [19] investigated the integration of BtG mechanisms with Attribute-based Encryption (ABE), which is a technique that uses public key cryptography to enforce fine-grained access control based on user attributes. The approach proposed in [19] is based on a hierarchy of emergency attributes employed to encrypt and decrypt data resources. Emergency attributes denote emergency severity levels activated / inactivated by a central authority, and are used to encode BtG policies. A BtG access is only possible when the emergency attribute required for decryption is active and the same attribute was active at encryption time.

Oliveira et al. [28] proposed a cloud enabled framework where a BtG mechanism is used to grant medical personnel the access to encrypted medical data managed by a cloud based application during emergency situations. Their framework revokes BtG accesses that are granted to the authorized medical personnel in the case the treatment is no longer needed.

Belguith et al. [13], proposed a BtG access control approach that leverages on: i) Shamir's secret sharing scheme [77], to derive secret shares from a secret access key, ii) ABE, used to encrypt the secret shares, and iii) QR encoding of the encrypted shares. In order to execute a BtG access users have to scan QR codes and recover individual keys with their attributes.

Tu et al. [81] propose a framework that provides the ABE-based access control in ordinary situations and the break-glass access control in emergency situations. The key point of the proposed framework is to move data decryption and the access control policy update tasks to the fog and cloud environment to reduce the computation on the user side.

Marinovic et al. [57], proposed a BtG model that employs a logic programming language to reason about unknown and conflicting information in policy decisions, and a policy specification language that allows security administrators to rule break-glass accesses. More precisely, their policy specification language does not assume that each contextual condition can be correctly established, thus, the security administrators can define the policies that contain decision gaps or conflicts by using this language. The authors also provide an enforcement model comprised of Break-the-Glass Policy Decision Point (BPDP), which evaluates the Break-the-Glass policies against access requests, and a Break-the-Glass Policy Enforcement Point (BPEP), which enforces authorization decisions.

Several BtG extensions have also been proposed for RBAC (e.g., [35, 58, 61]). In [35], the authors integrate the core RBAC model with BtG obligations that need to be performed before BtG access. Once a user requests to access a resource, their authorization framework makes one of these three decisions: grant the requested privilege normally, deny the requested privilege, and deny the requested privilege but grant break-the-glass privileges, if any policy related to the requested resource is configured with BtG obligations and if they are satisfied by users.

Nazerian et al. [61] introduced the concept of the emergency roles which are assigned to the users by their framework after the users request to have them in order to obtain additional privileges during emergencies. More precisely, under emergencies, a user makes a request to have a proper emergency role, and related obligations (e.g.,notifying the superior) for the emergency role must be performed by the user before obtaining the emergency role. The authors in [58] optimize the authorization framework proposed in [35] in terms of reducing memory and storage space for the BTG policies by employing a lightweight policy language (Ponder2) to define BTG policies.

The abovementioned papers have not been designed for IoT applications. Now let us introduce access control solutions for IoT environments that provide emergency management support based on the break-the-glass paradigm.

Yang et al. [87] propose a password-based break-glass access control mechanism for IoT-based healthcare ecosystems. The authors employ Attribute-based encryption (ABE) to handle access control in the ordinary situations. More specifically, each subject is assigned a secret key bound to his/her respective set of attributes and the patients specify access policies on their medical data under ordinary situations. In their framework, medical data of patients are encrypted and stored in the cloud. The subject employs his/her secret key to access encrypted medical data of the patient in ordinary situation, if his/her attributes satisfy the access policies defined by the patient. For emergency situations, the patient specifies a password and shares this password with the designated and trusted subjects who can extract the break-glass key from the shared password during emergency situations. In emergency situations, the break-glass key is employed to access encrypted medical data from the cloud and decrypt them. The same authors in [88] optimize their framework by finding and eliminating redundant and repeated encrypted medical data to save storage space in the cloud storage system. Another ABE-based approach for IoT-based healthcare ecosystems has been presented in [9], however, the authors propose a very similar approach as in [87] with only different implementation details.

Tasali et al. [79] propose an ABAC framework complemented with the Break-the-Glass features for medical IoT environments. In their framework, the BtG access can be requested per patient or the group of patients by the medical personnel and the BtG accesses do not have a predefined duration, once the medical personnel does not need the BtG accesses, he/she explicitly signals the end of the accesses.

We are also aware of an effort that offers a generic Break-the-Glass access control framework for IoT environments [82]. A key feature of [82] is the ability to detect emergencies from contextual information generated by IoT sensors. On prediction of an emergency, users are notified of the predicted situation, and contextually, the related break-glass policies are activated. The system then waits for possible break-glass requests.

3.2.2 Emergency Management Based on Emergency Policies

We are only aware of two approaches to emergency detection and data sharing regulation in emergency situations (i.e., [21,51]), based on emergency policies; however, none of them targets IoT ecosystems.

Kabbani et al. [51] proposed an approach to enforce ABAC policies in ordinary and emergency situations. In their framework, ordinary and emergency situations are detected employing a CEP based approach. However, the authors do not provide a systematic approach to gather events from event sources and to bind events to ordinary and emergency situations. Moreover, they do not experimentally evaluate the performance of their framework.

Carminati et al. [21] proposed a framework to enforce controlled information sharing under emergency situations, which employs a CEP system for emergency detection. In [21], emergency policies regulate the generation of temporary access control policies that override ordinary privileges in emergency situations. Once an emergency is detected, the applicable temporary access control policies are generated, stored in local repositories and kept active until either another emergency is detected or the current emergency is over.

We are also aware of a CapBAC framework [27] which provides access control and delegation mechanism within interconnected MQTT-based IoT environments. Their framework focus on data communication among the devices of a smart building and central emergency center after an incident happens in the smart building. The authors deploy an MQTT broker into the smart building environment and another MQTT broker outside the smart building, and establish a bridge connection between these brokers to allow remote users to access the shared data in the smart building. During an emergency incident, the local emergency management system, which is comprised of the IoT sensors deployed in the smart building, notifies the central emergency center, which is located in a remote environment. Then, the central emergency center delegates the required access permissions to data required by every rescue team which is located in a remote environment. The delegation mechanism relies on the delegation capabilities that show which users hold the rights to delegate their access permissions. While the general idea of this paper is interesting, the authors do not specify how the combination of IoT sensors is used to detect emergencies and how the central emergency center responds to different emergency levels.

The summary of the analysis throughout this chapter, which is given in Table 3.1, presents that most of the state-of-art access control solutions for IoT environments rely on centralized architectures. Since regulating data sharing across interconnected IoT environments has not been analyzed sufficiently in the literature (except for [37]), our access control framework introduced in Chapter 4 targets regulating data sharing across interconnected IoT environments by proposing a decentralized approach.

Moreover, to the best of our knowledge, none of the examined state-of-the-art access control for IoT environments throughout this chapter targets both emergency detection and regulating data sharing during emergency situation by enforcement of emergency policies. Our access control framework introduced in Chapter 5 targets regulating data sharing during the emergency and ordinary situations by enforcing emergency and ordinary policies respectively.

Table 3.1: Summary of the State-of-the-art review

Reference	Access Control Model	Architecture	Emergency Management
[42]	CapBAC	centralized	no
[27]	CapBAC	centralized	Emergency Policy approach without emergency detection
[46]	CapBAC	distributed	no
[14]	CapBAC	distributed	no
[86]	CapBAC	distributed	no
[60]	CapBAC	distributed	no
[56]	CapBAC	distributed	no
[49]	CapBAC	distributed	no
[84]	UCON	centralized	no
[52]	UCON	distributed	no
[29]	UCON	distributed	no
[43]	RBAC	centralized	no
[33]	RBAC	centralized	no
[11]	RBAC	centralized	no
[6]	RBAC	centralized	no
[22]	ABAC	centralized	no
[38]	ABAC	centralized	no
[30]	ABAC	centralized	no
[37]	ABAC	distributed	no
[55]	ABAC	distributed	no
[44]	ABAC	distributed	no
[31]	ABAC	distributed	no
[79]	ABAC	centralized	BtG approach
[87]	ABE based on ABAC	centralized	BtG approach
[88]	ABE based on ABAC	centralized	BtG approach
[9]	ABE based on ABAC	centralized	BtG approach
[82]	-	centralized	BtG approach

Chapter 4

Regulating data sharing across multiple Iot environments

4.1 Introduction

Internet of Things (IoT) applications are starting to be massively integrated into our lives [68] by providing advanced analysis services which leverage data generated by different IoT devices. Even though the benefits of these services are manifold, a trade-off between service utility and user privacy need to be considered [54], as these applications represent a potential threat to user privacy [68].

In the last years, research has coped with this issue by proposing several data protection solutions for IoT. In particular, for what access control is concerned, different approaches have been designed. Many access control solutions have been proposed for cloud-enabled IoT applications, which leverage on a centralized infrastructure that controls the communication and the activities of any node of an ecosystem (e.g., [4,5,34,41]). Other access control frameworks target IoT applications designed to control device behavior at the edge of the system [65]. However, the majority of the approaches in this category aims at regulating the access to data generated and exchanged within a single ecosystem (e.g., [22, 52]).

In contrast, an increasing number of IoT applications rely on IoT devices distributed across multiple ecosystems. For instance, Wang et al. [85] propose a collaborative edge computing framework for vehicular networks, which, by means of inter-ecosystem and intra-ecosystem collaborations, allows data sharing among network edges. Similarly, Zhang et al. [89] propose an edge computing framework, which, by means of virtual views of the data built by data owners for specific end-users, allow users of different ecosystems to share data.

Distributed architectures advocate parallel forms of data processing, and allow extending the scope of the sensed data to multiple ecosystems [83]. In addition, the independence from a centralized cloud-based infrastructure that remotely controls any aspect of an ecosystem favors better performance. However, along with these potential benefits, distributed approaches bring serious security threats, such as the possible disclosure of reserved data to unauthorized devices of other ecosystems.

In this chapter, we do a first step to address this issue, by proposing an ABAC framework to regulate data sharing among interconnected MQTT-based IoT ecosystems. Due to the pervasivity of many IoT scenarios, the proposed approach provides support for fine grained, contextual access control policies. For instance, let us consider the case of an Internet of Sport (IoS) (MyPersonalTrainer) which is introduced in Chapter 2. Simply, when deployed on exercise bikes, the MyPersonalTrainer app allows the gym frequenter to participate in cycling races, sharing data such as the current speed, and the covered distance. An access control policy that regulates data sharing across gyms may limit the exchange of rider performances to the race duration, granting the access only to the set of users registered for the race.

In addition, to enhance user control on the data generated by the administered devices, our framework allows users to specify their own preferences, which restrict the privileges granted by access control policies. For instance, a rider wishing to enforce a stricter privacy protection, may specify a user preference that, during a competition, filters out the identifiable data from the data shared with the riders of other gyms.

The framework proposed in this chapter extends [22], where an ABAC model has been proposed to control the communication of IoT devices operating in a single MQTT-based ecosystem. The proposed extension provides support to access control policies and user preferences that regulate data sharing among bridged MQTT-based ecosystems. Our framework also integrates an enforcement monitor, which regulates the data sharing, operating as an MQTT broker proxy that alters the communication flow between interconnected ecosystems. The monitor can be easily integrated into existing MQTT deployments, with basic configuration activities. The efficiency of the monitor prototype has been experimentally assessed, showing a reasonably low enforcement overhead in different testing scenarios.

To the best of our knowledge, the framework proposed in this chapter is among the earliest edge-based access control approaches that allow regulating data sharing across interconnected IoT ecosystems. We are only aware of another pioneering work by Carranza and Fong [37], which proposes an approach to regulate the interaction of message brokers of different ecosystems. However, [37] does not take into account context related information, operates at coarse grained level, and does not allow users to customize data sharing on the basis of their preferences.

The remainder of the chapter is organized as follows. Section 4.2 presents the adopted access control model, whereas Section 4.3 provides an overview of the proposed solution. Section 4.4 introduces the enforcement mechanism, and finally in Section 4.5 we present the experimental evaluation.

4.2 Access control across different MQTT environments

To manage data sharing across different IoT ecosystems, we enhance the model in [22], with the ability to regulate message exchange among MQTT publishers and subscribers possibly belonging to different ecosystems. As a reminder, we have introduced the MQTT protocol in Chapter 2.1.

In what follows, we refer to a scenario where two MQTT-based ecosystems, respectively referred to as *local* and *remote*, communicate to each other by means of *interconnected brokers*. We denote two brokers as interconnected when one of them has been configured as bridging broker and specifies the other one as connection target.

The main differences with previous model version are related to subject, access control policies and user preferences.

A subject s can denote : i) a client of the local ecosystem who connects to the local broker with the aim to publish or receive messages, possibly on behalf of a user,[1] ii) a local broker of one ecosystem which aims at exchanging application messages with a remote ecosystem, or iii) a bridging connection through which a bridging broker communicates with a remote broker. A local broker may forward messages published by its local clients to a remote broker, as well as receive messages published in a remote ecosystem which then will be forwarded to the rightful subscribers of its ecosystem.

As in [22], access control policies are specified to grant subjects the read or write access to messages referring to a set of topics. Like in [22], if the subject is a client, the read privilege represents the right to receive messages on a subscribed topic, whereas the write authorization specifies the right to publish a new message. In contrast, if the subject is a broker or a bridging connection, a read authorization represents the right to receive messages that have been published in a remote ecosystem, whereas a write authorization specifies the right to forward messages to a remote ecosystem. Policies granting read/write privileges to a broker are specified by the security administrators of the ecosystem that hosts the broker.

As we explained in Def.2.1 at Chapter 2.2.1, an access control policy p is a tuple $\langle$*s, tf, exp, pr*$\rangle$, where *s* refers to the subjects constrained by *p*, *tf* specifies a topic filter expression, *exp* is a parametric predicate, whereas *pr* specifies the *read /write* privileges granted to s if *exp* is satisfied. This definition is still applicable to formalize the concepts of access control policy.

The only difference is that component s of p in Def. 2.1 can also refer to a broker, or to a specific connection of a bridging broker. If s refers to a broker, the read/write privilege granted by p models the broker privilege to receive/forward messages. If s refers to a bridging connection, the granted privilege applies to the bridging broker that handles the connection specifying the broker communication ability for this connection.

Example 4.1. *Let us now focus on policies regulating brokers interaction specified for our running case. A policy* p_2*=*$\langle$*MyGym, +/performance/ts/+, isOpeningHour(t), rw*$\rangle$ *authorizes MyGym's broker to forward and receive messages encoding performance data of runners attending a training session ts during MyGym's opening hours. Since no bridging connection is explicitly referred, the privilege applies to any connection. A similar policy,* p_3*, can be specified for the RemoteGym's broker as* $\langle$*RemoteGym, +/performance/ts/+,*

[1]MQTT allows specifying a user name within CONNECT control packets (see Table 2.1), with the aim to support authentication mechanisms. However, user names are not mandatorily specified.

isOpeningHour(t), rw⟩. In contrast, a less restricting policy can be specified for the Analyzer's broker as p_5=⟨Analyzer, +/performance/#, true, r⟩. This policy authorizes the Analyzer's broker to receive, from any MyGymBrand's gym, messages on topics referring performance data of any training session.

Users can further constrain the access to messages published on their behalf, through user preferences. In the extended model, user preferences can either restrict the read access to published data by clients, or brokers forwarding privileges.

For instance, users may specify user preferences requiring that their running data are only shared within the sport hall where data have been sensed, without being tracked, analyzed, or forwarded to frequenters of other gyms.

In order to allow users to restrict brokers forwarding privileges, the user preference definition (see Def. 2.2 in Chapter 2.2.1) has been enhanced with an additional component, denoted *bp*, which allows specifying the forwarding preference. *bp* may refer to: i) the name of a target ecosystem *te*, if an user preference *up* constrains the forwarding to *te* only, ii) *, if *up* constrains the forwarding to any remote ecosystem, or iii) $\perp$, if *up* applies to the read access to the referred messages by subscriber clients. On the basis of *bp*, the parametric predicate *exp* specifies a precondition: i) to the forwarding of a protected message to a remote ecosystem, or ii) to the receiving of the message by rightful subscribers.

Example 4.2. *Now let us reconsider the user preference up_1 introduced in Example 2.5 that exemplifies the user preference in the preliminary work [22]. As a reminder, in Example 2.5 Mary, who is a frequenter of MyGym, limits to Alice, who is a coach of MyGymBrand, the privilege to access Mary's performance data related to the training session ts by specifying user preference up_1.*

Now in our current access control model the same user preference can be redefined as: up_1=⟨Mary, +/performance/ts/+, $\perp$, s.rid="coach"$\wedge$ s.uid="Alice"⟩. It is worth noting that during Mary's training session, depending on the work shift, Alice may be working within MyGym or RemoteGym, however, the effect of up_1 is independent from the gym where Alice works when Mary is training. In addition, Mary, who does not agree to be tracked by the analysis server, specifies the preference up_2=⟨Mary, +/performance/ts/+, Analyzer, false⟩, which prohibits the sharing of her training session data with that server. Let us also consider the user preference up_3=⟨Bob, +/performance/ts/+, RemoteGym, false⟩, specified by Bob, a frequenter of MyGym. up_3 prohibits the forwarding of Bob's performances during the training session ts to RemoteGym.

User preferences restrict the privileges which are granted by access control policies, constraining the reading privileges of subscribed clients, and the broker ability to forward messages to a broker of a different ecosystem. More precisely, a subscriber subject s of a given ecosystem re can read a message m published on a topic tp by a publisher subject ps operating within re on behalf of a user u iff within re: i) there exists an access control policy p that grants s read access to tp and ii) there exists at least one user preference up, among those specified by u with a topic filter expression that is matched by tp, which grants s read access to tp. In case s and ps belong to different ecosystems, additional checks are required to allow the forwarding of m from the ecosystem of ps to the one of s. Let us denote with se and pe the ecosystems of s and ps, respectively, whereas b_{pe} and b_{se} denote the brokers of these ecosystems. b_{pe} can forward to b_{se} a message m published on a topic tp by a subject ps of pe on behalf of a user u iff within pe: i) there exists an access control policy p_w that grants b_{pe} the write access to tp, and ii) there exists at least one user preference up among those specified by u with a topic filter expression that is matched by tp, which grants b_{pe} write access to tp. A message m forwarded by b_{pe} can be received by b_{se} iff within se there exists at least one access control policy p_r that grants b_{se} read access to tp.

Example 4.3. *Let us suppose that Mary and Bob are registered to a course of MyGym, and that they are attending the training session ts, which, within RemoteGym, is coached by Alice. Let us assume that Bob and Mary are running on the tapis roulants tpr_1 and tpr_2, which have been configured to share performance data of all runners attending the training session.*

Let us consider a policy p_1 which grants frequenters enrolled to a gym course the privilege to publish performance data and read performance data of other trainees. p_1 can be specified as: ⟨frequenter, +/performance/ts/+, isEnrolled(s.sid),rw⟩, where isEnrolled(s.sid) is a function that checks whether the subject s.sid is enrolled to a gym course.

The publishing and receiving of data is regulated by policy p_1, which grants any frequenter that is enrolled to a gym course and is attending a training session ts, the privilege to publish messages specifying his/her performances, and to receive messages over topics specifying performance data of other runners attending ts. The publishing by tpr_1 and tpr_2 of messages referring Mary and Bob performances during ts, complies with p_1, therefore, both tpr_1 and tpr_2 are authorized to publish. However, the user preference up_1 specified by Mary, constrains the receiving of messages published by tpr_1 (see Example 4.2), as no frequenter or coach, but Alice, can access Mary's data. Therefore, on the basis of the applicable policies and preferences, during ts, Mary can see Bob's performance, whereas Bob cannot see Mary's data. According to policy p_2 (see Example 4.1) performance data published by tpr_1 and tpr_2 can be forwarded by MyGym broker to any remote ecosystem. However, the forwarding of Mary's and Bob's data is constrained by the user preferences up_2 and up_3 (see Example 4.2). More precisely, due to up_2, Mary's data cannot be sent to the Analyzer's broker, however, no preference prohibits the forwarding to RemoteGym. In contrast, on the basis of up_3, Bob's data cannot be forwarded to RemoteGym, but no restriction has been specified for Analyzer. The receiving of Mary's forwarded messages by the RemoteGym's broker is regulated by policy p_3 (see Example 4.1), which authorizes, during the opening hours of RemoteGym, the receipt of messages over performance related topics referring to the training session ts, which have been forwarded by MyGym's broker. Similarly, according to p_5, Analyzer can receive messages over performance related topics of any training session which have been forwarded by MyGym's broker, thus it can also receive Bob's data.

Once the forwarded message m is received by b_{se}, m can be dispatched to the rightful subscribers of se. More precisely, m can be accessed by a subscriber subject s of se iff: 1) within *se* there exists at least one access control policy p_r which grants s the read privilege to messages on topics that match a topic filter expression that is also matched by tp, and 2) the access complies with at least one of the user preferences specified by the original publisher of m within pe.

Example 4.4. *Let us now focus on the routing, within the RemoteGym ecosystem, of messages on performance related topics, which have been originally published by tapis roulant* tpr_1 *on behalf of Mary, and then forwarded to the RemoteGym ecosystem by MyGym's broker (see Example 4.3). Alice's app, has been configured to subscribe the receiving of performance data published by any device. However, the receipt is regulated by the applicable access control policies and user preferences. More precisely, the receiving requires the joint satisfaction of the policy p2 (see Example 4.1), which grants Alice the privilege to receive messages on performance related topics during her working shift, and the user preference* up_2 *(see Example 4.2), specified by Mary, which grants the access to Alice only. Due to* up_1*, no client within MyGym and RemoteGym is authorized to access Mary's data, but those requesting the access on behalf of Alice.*

4.3 Overview

Now we present an overview of architectural aspects of the proposed framework. We target an application scenario where two MQTT-based ecosystems, each composed of multiple MQTT clients and a local broker. Inter-ecosystem communication is achieved by configuring the local broker of one of the two ecosystems as a bridging broker specifying the other broker as connection target.

The proposed framework enhances the approach introduced in [22], which targeted the regulation of message flow within a single ecosystem, with the ability to regulate the communication of MQTT clients of different IoT ecosystems. More precisely, the enforcement monitor proposed in [22], denoted in what follows as *local* monitor, has been enhanced to enforce user preferences possibly specified within a different ecosystem. Additionally, a new enforcement monitor, denoted as *bridging monitor*, has been designed, which, on the basis of access control policies and user preferences, regulates message passing between interconnected brokers.

The proposed approach requires to interpose the local monitor between the local clients and the respective brokers, and the bridging monitor in between the brokers of the interconnected ecosystems. More precisely, 1) the clients of each ecosystem are configured to connect to the respective monitors rather than directly to their brokers, 2) the local monitors are connected to the respective local brokers, 3) the local broker, which has been configured as a bridge, specifies the bridging monitor as connection target, and 4) the bridging monitor specifies the local broker of the other ecosystem as connection target. As such, the local monitors of the connected ecosystems behave as proxies of the respective brokers, and are the only components that can communicate with the brokers on behalf of their clients. In contrast, the bridging monitor behaves like a proxy of the broker referred to as connection target of the local bridging broker, and is the only component enabling

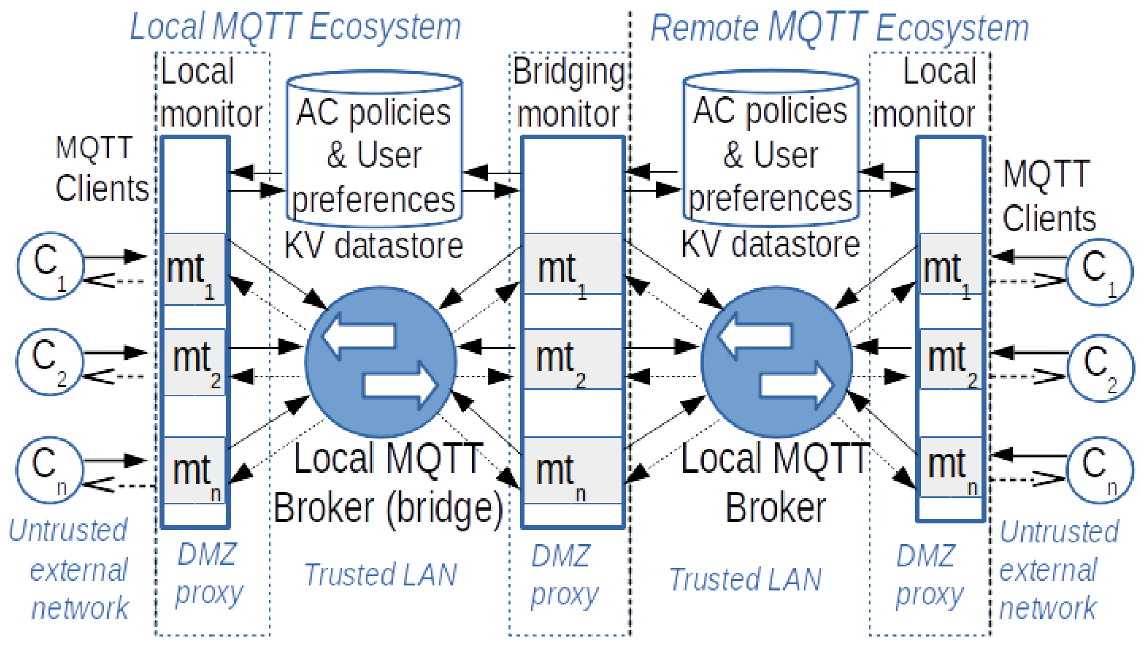

Figure 4.1: A high level view of the system architecture in [24]

the communication between different ecosystems. Figure 4.1 presents an high level view of the system architecture. The designed approach does not require ad-hoc implementations of clients and brokers, and it is independent from specific client and broker versions. The local and bridging monitors can also operate within heterogeneous environments, where different versions of clients and brokers cooperate.

According to the system architecture shown in Figure 4.1: 1) local MQTT brokers are deployed in different trusted LANs, whereas 2) MQTT clients in untrusted external networks, 3) whereas the local monitors and the bridging monitor are hosted by DMZ proxies at the LAN interfaces. Firewalls placed at the interfaces of these three DMZ proxies are configured to prohibit unmediated connections with the local brokers.

The activities of the local and bridging monitors are subject to access control policies and user preferences (cfr. Section 4.2) regulating the communication within each ecosystem, and between interconnected ecosystems. Access control policies and user preferences are handled by key-value datastores, deployed in the interconnected ecosystems. The policy set handled by each datastore allows regulating the internal communication in the respective ecosystem, as well as the messages exchanged with the other ecosystem. The bridging monitor accesses the policy sets of both datastores, whereas the local monitors only the set in the datastore of the respective ecosystem.

4.4 Enforcement

In order to ensure that the internal message flows, the local monitors analyze and possibly alter the flow of MQTT control packets exchanged by their clients, whereas the bridging monitor performs similar operations on the flow of packets exchanged by the brokers of the two ecosystems. In the remainder of this section, we first present the rationale of the enforcement mechanism, and then we focus on its specification details.

4.4.1 Enforcement rationale

The message passing within a target ecosystem is regulated by instances of the enforcement monitor proposed in [22] and summarized in Section 2.2.2, which have been enhanced to enforce user preferences possibly specified within a different ecosystem. The enhanced version of the monitor differs from the original one for the management of security metadata, instrumental to access control. Indeed, different from [22], during the analysis of client publishing requests, along with the user preferences, the monitor embeds into the analyzed packet's payload the subject, object, and environment attributes that model the context within which the publishing request has been issued. Similarly, during the analysis of publishing requests the monitor extracts all these attributes along with user preferences from the packet's payload, deriving the context in which: i) the message has been originally published, and ii) should be received by the subscriber. The monitor evaluates the embedded preferences wrt all derived attributes. Finally, before issuing a message to a rightful and authorized subscriber, the monitor removes from the packet's payload all previously embedded metadata.

Let us now focus on the communication between ecosystems, which is regulated by the bridging monitor. By assumption, the local broker of one of the two ecosystems has been configured as a bridging broker specifying the bridging monitor as connection target.

Message sharing between the two ecosystems is subject to the bridging rules that configure the communication abilities of the local brokers (cfr. Chapter 2). Hereafter, we refer to bridging rules whose component *dr* has been set to *out* as *output bridging rules*, and to *in* as *input bridging rules*. Output bridging rules specify the topics of the messages published within the local ecosystem that can be forwarded to the broker of the other ecosystem, and the remapping criteria with which the topic of a message is modified before the message is actually forwarded, whereas input bridging rules specify the topics of the messages published within the connected ecosystem which the local broker subscribes to receive, and the criteria with which the topic of the received messages is remapped before they are forwarded to a rightful local subscriber. As a first step, the local broker of the ecosystem where the bridging monitor is hosted sends a connection request to the bridging monitor, which in turn forwards the request to the local broker of the other ecosystem, as it was the original sender of the request. If the request is accepted, a communication channel connecting the two local brokers is open, which is controlled by the bridging monitor. In what follows, we denote as *connecting broker* the local broker that sends the connection request, whereas the other is denoted as *target broker*. Through the established channel, the connecting broker sends a subscription request for any input bridging rule,[2] and then waits for control packets issued by local clients and by the bridging monitor.

The bridging monitor has been designed to forward any control packet received by the local brokers to the respective receiver, except packets encoding publishing requests which require additional operations.

Let us first consider publishing requests originating from the ecosystem managed by the connecting broker. For any received publishing request issued by a local client that matches an output bridging rule, the connecting broker remaps the topics of the packet (see Chapter 2) and forwards the request to the bridging monitor. On the packet reception, the bridging monitor extracts from the payload: 1) the user preferences specified by the subject who has originally requested the publishing, and 2) the attributes that model the publishing request context.[3] Then, the bridging monitor selects from the extracted preferences those constraining message forwarding to the other ecosystem, and evaluates them wrt the derived attributes. If no preference is satisfied, the message is blocked. Otherwise, if at least one preference is satisfied, the bridging monitor enforces the applicable access control policies. More precisely, it selects from the policy set of the ecosystem managed by the connecting broker those policies regulating the forwarding of messages to external ecosystems, and from the policy set of the ecosystem managed by the target broker those policies regulating the receiving of messages from external ecosystems. If the forwarding complies with at least one policy of both sets, the message is sent to the target broker, otherwise the forwarding is forbidden. In contrast, the target broker, upon receiving a publishing request from the bridging monitor, forwards, on the basis of the message topic, a copy of the received packet to communication channels that connect the broker to the rightful subscribers of the other ecosystem, each controlled by the local monitor. For any message receiver candidate, the monitor checks whether the considered subscriber, on the basis of the preferences in the message payload and the applicable access control policies, is authorized to receive the message, and, in this case, it removes all metadata from the payload and forwards the packet.

Similarly, the target broker forwards to the bridging monitor any publishing request received from a local publisher that matches a previously subscribed topic filter.[4] The bridging monitor handles the publishing request with an approach symmetric to the one just described for the opposite message flow. The only difference is related to the local broker management of publishing requests from the bridging broker. Indeed, due to the remapping criteria specified by the input bridging rules, the topic of the message is remapped before the message is forwarded to the channels, controlled by the local monitor, which connect the broker with rightful local subscribers.

[2]Each subscription request specifies as topic filter the concatenation of the remote prefix (see Section 2.1) and the topic filter of the considered input bridging rule.

[3]Such data have been added to the payload by the local monitor.

[4]We remind that any subscription is achieved on behalf of the local broker on the basis of an input bridging rule.

4.4.2 The enforcement mechanism in details

We start to consider configuration aspects enabling broker to broker communication within the system architecture introduced in Section 4.3, which are instrumental to the proposed enforcement mechanism.

Let us hereafter refer to the bridging monitor as *bm*, and let us denote as *lb* the local broker that has been configured to operate as bridging broker specifying the bridging monitor *bm* as connection target, whereas we denote with *rb* the broker of the other ecosystem.

On behalf of local clients, and on the basis of the configured bridging rules (cfr. Chapter 2), *lb* can publish messages to *rb*, and can subscribe the receiving of messages published within *rb*'s ecosystem.

The interaction of *lb* and *rb* starts with a connection request cp_{CN}, sent by *lb*. Upon receipt of cp_{CN}, the bridging monitor *bm* extracts the credentials of *lb* from the *CONNECT* control packet, and forwards the packet to *rb*. *rb* authenticates the subject and replies with a *CONNACK* control packet cp_{CA}, which is received by *bm* and then forwarded without any modification to *lb*. cp_{CA} specifies whether the connection request has been accepted or refused by *rb*, and, in the latter case, the cause. If cp_{CA} encodes the acceptance of the connection request, *lb* can start communicating with *rb*.

Once the connection has been established, for any input bridging rule *ibr* that has been specified for *lb*, *lb* sends a *SUBSCRIBE* control packet cp_{SB} to *rb*, specifying as topic filter the expression resulting from the concatenation of the components *rp* and *tp* of *ibr*. *bm* receives cp_{SB} and forwards it to *rb*, which keeps track of the topic filter *rp+tf*, and sends back a *SUBACK* packet notifying the receiving of the request. Once received by *bm*, the acknowledgement is forwarded to *lb*, which, on the basis of the specified bridging rules, is now ready to forward messages published by its local clients to *rb*, as well as to receive messages published in the *rb*'s ecosystem.

We now focus in more details on the enforcement mechanism implemented by the bridging monitor, whose joint work with the local monitors allows regulating the communication of clients belonging to different ecosystems.

Let us start to consider the forwarding of messages published within *lb*'s ecosystem. The payload of any publishing request cp_{PB} received by *lb* includes: 1) the user preferences, if any, specified by the user, on behalf of whom the message has been published, and 2) the subject, object, and environment attributes which model the context within which the publishing request has been issued. If the topic *tp* referred to by cp_{PB} is matched by an output bridging rule *obr* of *lb*, *lb* remaps the topic. prepending to *tp* the remote prefix *rp* of *obr*. The resulting control packet, referred to as $cp_{PB''}$, is then forwarded to *bm*, which, by assumption, has been specified as connection target of *lb*. Upon the receipt of $cp_{PB''}$, *bm* extracts from the packet's payload the user preferences and the subject, object, and environment attributes. For any included user preference *up*, *bm* checks whether *up* constrains the forwarding of $cp_{PB'}$ to the ecosystem managed by *rb*, and, in such a case. evaluates the parametric predicate *exp* of component *bp* of *up* (see Section 4.2) with respect to the extracted subject, object, and environment attributes. If the predicate of at least one of the user preferences that constrain the forwarding is satisfied, *bm* checks whether there exists at least one access control policy among those specified in the ecosystem of *lb* that authorizes the forwarding, and at least one in the policy set of the ecosystem of *rb* that authorizes the import, otherwise the forwarding is blocked. The selection of the applicable policies in both ecosystems is achieved by matching the topic filter of any policy *p* specified for *bm*, with the topic of $cp_{PB'}$. Policies are evaluated with respect to the attributes extracted from the payload. If the parametric predicate of at least one policy is satisfied, *bm* forwards $cp_{PB'}$ to *rb*, which in turn sends $cp_{PB'}$ to its local monitor. The local monitor then handles the publishing request as described in Section 2.2.2. The approach used by the bridging monitor to regulate the forwarding of messages published within the ecosystem managed by *rb* is similar to the previously explained one. The only difference is related to the topic remapping task operated by *lb*. Indeed, on receipt of a publishing request cp_{PB} from the bridging monitor *bm*, *lb*, on the basis of the specified input bridging rules, first redefines the topic of cp_{PB} removing the remote prefix, and then forwards the packet to the local monitor which will handle the request (see Section 2.2.2).

4.5 Performance Analysis

In this section, we first shortly present core aspects related to the implementation of the bridging monitor, and than we evaluate the efficiency of the proposed enforcement mechanism with two experiment sets.

4.5.1 Implementation

We now shortly discuss implementation aspects of the bridging monitor. [5] A high level view of the bridging monitor architecture and of the related control flow is shown in Figure 4.2. The bridging monitor *bm* has been designed to listen for connection requests from *lb*.[6] Upcoming connections requests are handled by a connection handler, which, on receipt of a new request from *lb*, opens a communication channel $\mathrm{cc}_{\mathrm{lb}}$ with the local broker, and one channel $\mathrm{cc}_{\mathrm{rb}}$ with *rb*, the local broker of the other ecosystem. The handler also instantiates a monitoring task *mt*, which regulates the flow of control packets flowing through $\mathrm{cc}_{\mathrm{lb}}$ and $\mathrm{cc}_{\mathrm{rb}}$. A pair of message queues are used to keep track of the messages flowing as input and output to the bridging monitor through these channels, respectively denoted $\mathrm{in}_{\mathrm{lb}}$, $\mathrm{out}_{\mathrm{lb}}$, $\mathrm{in}_{\mathrm{rb}}$ and $\mathrm{out}_{\mathrm{rb}}$.[7] The monitoring task enqueues packets intended for *lb* and *rb* to $\mathrm{out}_{\mathrm{lb}}$ and $\mathrm{out}_{\mathrm{rb}}$, whereas draws packets issued by *lb* and *rb* from $\mathrm{in}_{\mathrm{lb}}$ and $\mathrm{in}_{\mathrm{rb}}$, respectively. A pipeline of packet handlers is used by *mt* for the marshalling of output packets, as well as for the unmarshalling of packets to be added to the input queues. Any control packet *cp*

[5]The local monitors are extended version of the enforcement monitor presented in [22]. The interested reader can refer to [22] for a detailed description of implementation aspects related to these monitors.

[6]We remind that *lb* denotes the local broker that has been configured as a bridging broker specifying *bm* as connection target.

[7]*in* and *out* denote the verse of the flow from the monitor perspective, whereas *lb* and *rb* denote the broker associated with the queue.

received as input is handled by *mt*, on the basis of the enforcement mechanism introduced in Section 4.4.2. Therefore, on receipt of a publishing request from *lb* and *rb*, the applicable policies are derived and checked. Such policies are stored within kvd_{lb} and kvd_{rb}, the key value datastores of the connected ecosystems.

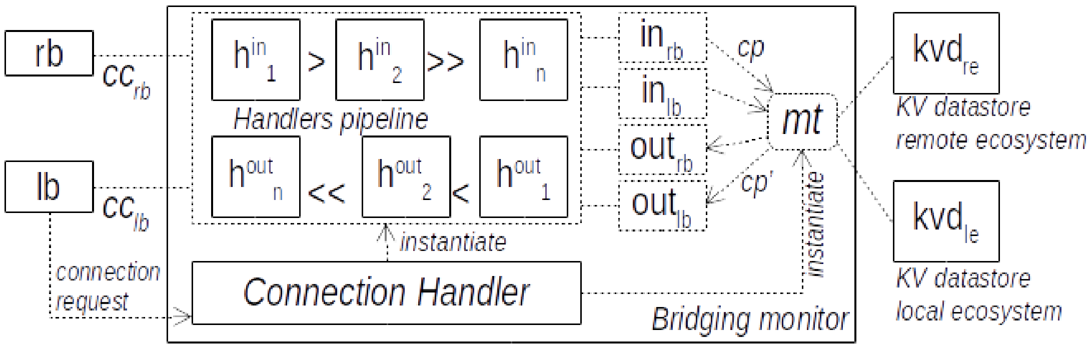

Figure 4.2: Bridging monitor architecture

The proposed framework reuses the same criteria adopted in [22] for the modeling of access control policies and user preferences within the datastores, as it has been shown that these favor a very efficient execution of the queries that derive all per request applicable policies and preferences. The interested reader can refer to [22] for more details.

4.5.2 Experiments

In this section, we experimentally evaluate the efficiency of the proposed enforcement mechanism with two experiment sets. Our experiments refer to an application scenario characterized by two MQTT based interconnected ecosystems. One of the local brokers has been configured to operate as a bridge that considers the other broker as a connection target. Brokers are instances of Mosquitto v.1.4.10[8], a popular MQTT broker. We assume that MQTT clients operating within the local ecosystems are handled by 100 users, each managing 1 to 4 clients. Clients are distributed between the two ecosystems in such a way that any user who handles a client in one ecosystem cannot handle a client in the other one. Clients distribution is carried out on the basis of three *configurations*. In the first configuration, the target ecosystem only includes publisher clients, whereas the connecting one only subscribers. In the second configuration, the distribution criterion is inverted, whereas, in the third one, publishers and subscribers are equally distributed between the two ecosystems.[9] For each configuration, we consider three *deployment options*, each involving a different number of clients. The first option refers to a client set of 20 publishers and 20 subscribers, the second to 100 publishers and 100 subscribers, whereas the third one involves 200 publishers and 200 subscribers. To more easily make reference to the

[8]https://mosquitto.org/

[9]We remind that the connecting ecosystem is the one hosting the bridging broker, whereas the other one is the target ecosystem.

tested deployment settings, we refer to them as *scenarios* and we use the notation $S_{i,j}$, where $i, j \in [1, 2, 3]$, to denote the scenario corresponding to the i-th deployment option of the j-th configuration. For instance, $S_{2,3}$ refers to a scenario where 100 publishers and 100 subscribers are equally distributed between the two ecosystems. The considered scenarios are summarized in Table 4.1.

Table 4.1: Testing scenarios

	Connecting ecosystem		Target ecosystem	
Scenario	#publishers	#subscribers	#publishers	#subscribers
$S_{1,1}$	0	20	20	0
$S_{2,1}$	0	100	100	0
$S_{3,1}$	0	200	200	0
$S_{1,2}$	20	0	0	20
$S_{2,2}$	100	0	0	100
$S_{3,2}$	200	0	0	200
$S_{1,3}$	10	10	10	10
$S_{2,3}$	50	50	50	50
$S_{3,3}$	100	100	100	100

In the considered deployments, publisher clients have been configured to send 1 publishing request per second, whereas a single subscription request is issued by each subscriber client. Moreover, clients have been configured to issue publishing and subscription requests specifying all the same Quality of Service level (QoS).

The routing activities of the bridging broker are regulated by the bridging monitor (cfr. Figure 4.1). This monitor intercepts and possibly alters the flow of the messages exchanged by the brokers on the basis of the access control policies and user preferences which have been specified within the local ecosystems. The specified access control policies, 80 per ecosystem, have been defined in such a way that, 40 policies constrain the local brokers ability to forward messages to the other ecosystem, whereas the remaining 40 constrain the local brokers ability to receive messages published in the other ecosystem. User preferences have been defined in such a way that each user specifies at most one preference which constrains the corresponding ecosystem's broker ability to forward messages to the other ecosystem.

Our experiments aim at assessing the enforcement overhead, by measuring the *transmission time*, namely the time spent by a message published in one of the ecosystems to reach a rightful subscriber, and the packets *throughput*, considered as the number of control packets which are handled per second.

Experiment 1 Our first set of experiments consider a scenario where neither of the two ecosystems is equipped with an enforcement monitor, therefore local clients are directly connected to the respective brokers. The bridging monitor and the clients are hosted by desktop PCs equipped with i7 64-bit Quad Core CPU and 16 GB of RAM, whereas the local brokers by Raspberry Pi 3 Model B devices (equipped with a 64-bit Quad Core CPU and 1 GB of RAM).

In order to analyze the enforcement overhead, for each scenario, we compare the transmission time measured in a deployment devoid of the bridging monitor, with a deployment where the monitor is active. Two cases per scenario are considered, where all clients issue their publishing and subscription requests specifying QoS 0 and 2, respectively.

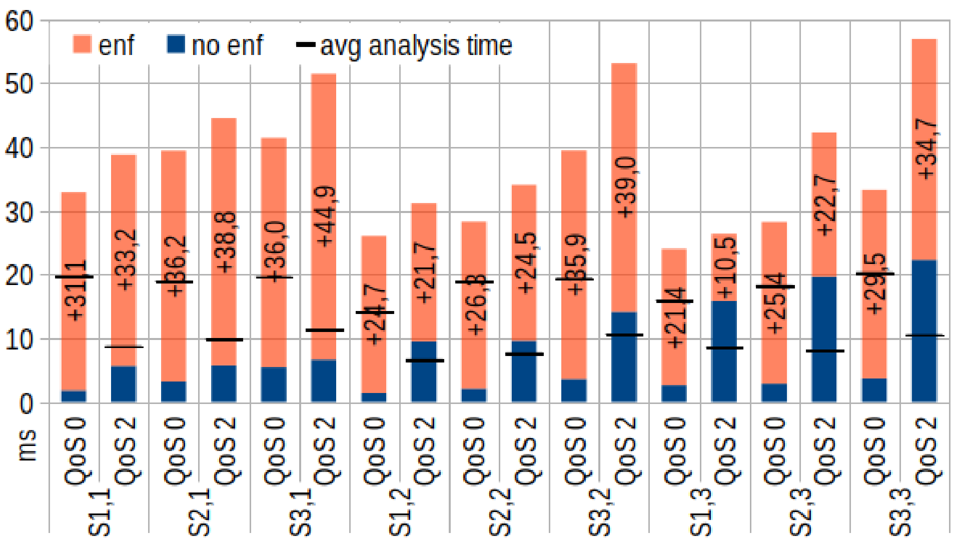

Figure 4.3: First experiment: transmission time analysis

Figure 4.3 shows the transmission time and the overhead measured for each considered case, as well as the average analysis time of control packets issued by local brokers. The lower part of each bar shows the transmission time related to deployments lacking the bridging monitor, whereas the upper part shows the transmission time in deployments where the monitor is active, and the corresponding time overhead. The average time (per control packet) is represented by horizontal lines overlying the transmission time bars.

Overall the time overhead related to cases with QoS 2 is always below 45ms, whereas the one related to QoS 0 is below 37ms, showing reasonably good performances of the bridging monitor. This behavior is due to the number of control packets that are exchanged per single publishing request, which, with QoS 2, is higher than with QoS 0. As a matter of fact, when QoS 2 is specified, for each publishing request, the local brokers also exchange the control packets cp_{PRC}, cp_{PRL}, and cp_{PC}. However, the analysis of these control packets requires less time than the analysis of publishing requests (i.e., cp_{PB}, namely the only ones involved in measuring cases specifying QoS 0), as these packets, once intercepted and recognized by the monitor, are directly forwarded to the respective broker. The low processing time of these packets lowers the average analysis time of each measuring case. For this reason, for each scenario, the transmission time related to QoS 2 measuring case is higher than QoS 0 case, whereas the trend is inverted for the measured average analysis time. This scheme can be observed with any analyzed scenario. The lowest transmission times have been measured in scenarios referring to the first deployment option (e.g., $S_{1,3}$).

The times grow in scenarios that refer to the second deployment option (e.g., $S_{2,3}$) and even more in scenarios referring to the third option (e.g., $S_{3,3}$). This behavior is due to the number of clients involved, and the volume of messages that need to be handled by the bridging broker and the bridging monitor. In contrast, the comparison of scenarios referring to the same deployment option but different configurations show negligible time variations.

For each considered case, the red bar in Figure 4.4 shows the throughput of the bridging monitor, whereas the blue bar the throughput of the broker in deployments devoid of the monitor. The trends in Figure 4.4 are similar to those observed for the transmission time analysis.

Due to the analysis of policies and preferences that regulate the transit of any control packet, the rate of packets that is handled is lower than in scenarios with no enforcement mechanism. However, the observed rates combined with the previously considered transmission times (see Figure 4.3) appear as a reasonably good result.

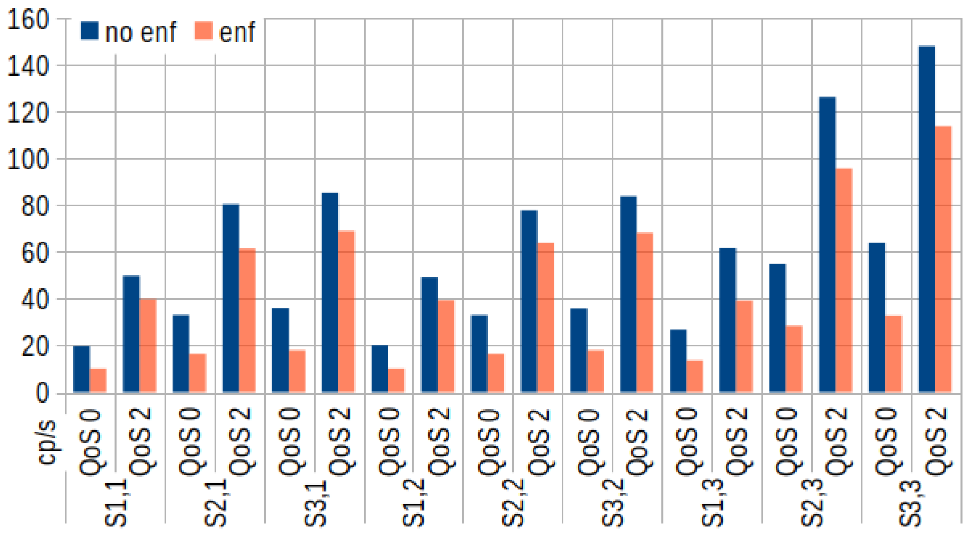

Figure 4.4: First experiment: throughput analysis

Experiment 2 In the second set of experiments each of the interconnected ecosystems also includes an enforcement monitor that regulates the ecosystem internal communication. Therefore, the considered system architecture corresponds to the one shown in Figure 4.2 and discussed in Section 4.3. The bridging monitor and the clients are hosted by desktop PCs, equipped with i7 64 bit Quad Core CPU and 16 GB of RAM, whereas the MQTT brokers of the two ecosystems are hosted by two Raspberry Pi 3 Model B devices (64 bit Quad Core and 1 GB RAM), which also host the local monitors.

In this second set of experiments, we aim at assessing the enforcement overhead introduced by the combined work of the local and bridging monitors.

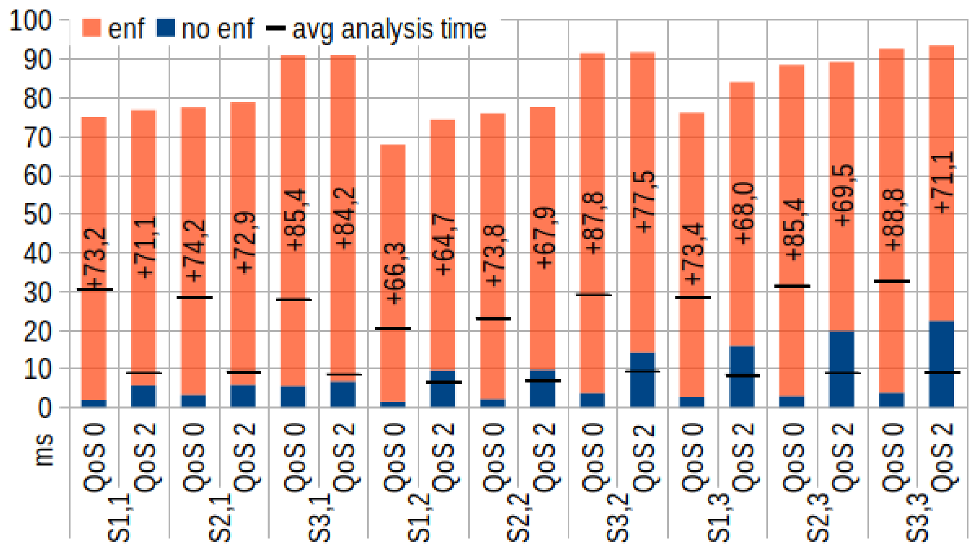

Figure 4.5: Second experiment: transmission time analysis

The diagram in Figure 4.5 shows, for each considered case, the transmission time related to a deployment devoid of enforcement monitors, which is matched against the transmission time measured in a deployment where all monitors are active.

Overall the transmission time is always below 94ms, whereas the time overhead always below 89ms. Therefore, the addition of the local monitors causes a reasonably contained growth of the transmission time and the time overhead wrt the first set of experiments. The same trends that have been observed in Figure 4.3 are also visible in Figure 4.5.

Finally, the diagram in Figure 4.6 shows the measured throughput. For each case, the red bars show the throughput of the bridging monitor, whereas the blue bars show the throughput of the broker in deployments devoid of the monitor.

The trends visible in Figure 4.6 are aligned with the ones observed with the first set of experiments (see Figure 4.4). However, in this case, due to the filtering operated by the joint work of the local monitors, a lower ratio of control packets per second reaches the bridging monitor (cfr. Figure 4.4).

Overall, the experiments show an enforcement overhead that is always reasonably contained, even in scenarios where data sharing across ecosystems is regulated by three monitors.

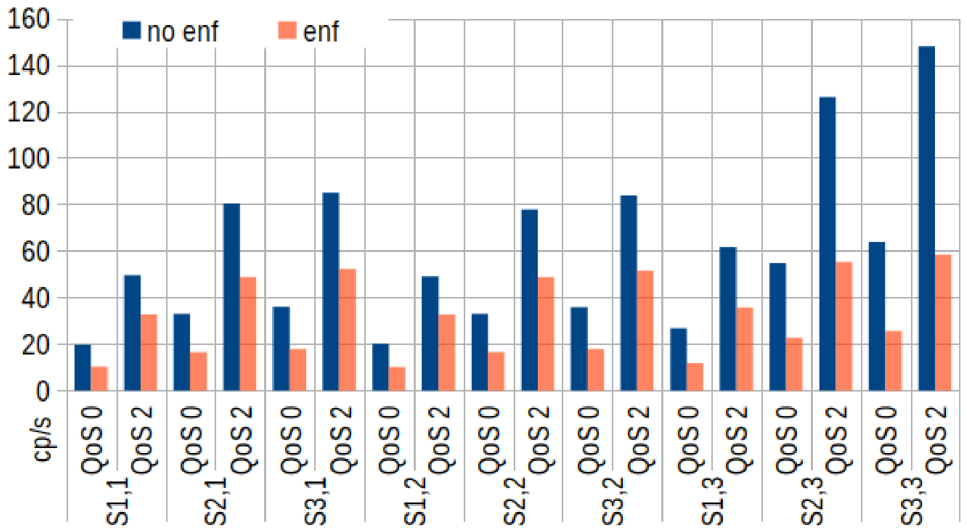

Figure 4.6: Second experiment: throughput analysis

Chapter 5

Regulating data sharing under emergencies

5.1 Introduction

An emergency is a critical situation that happens suddenly and requires prompt management to avoid harmful results. Recent emergencies, such as the COVID-19 pandemic, have shown that, due to scarcely information sharing, emergency protocols often fail in fully achieving their goals. For instance, during the COVID-19 pandemic, contact tracing has been pointed out by the World Health Organization as a strategic tool for contrasting SARS-CoV-2 diffusion and reducing COVID-19 mortality [62]. However, manual contact tracing methods proved scarcely applicable, highly demanding in terms of time and human resources, and overall impractical with a high number of new daily cases. Moreover, people's ability and willingness to derive and disclose sensitive information, as visited places and persons met, have further hindered their application and efficacy. Contact tracing apps have addressed the scalability and performance issues of manual methods. However, due to a scarce perception of the enforced data protection, in several western countries, citizens proved unavailable to install and use these apps [2]. As a consequence, the efficacy of contact tracing has been undermined by limited population coverage. These facts suggest that efficient data sharing is a key requirement for emergency management, and should be complemented with proper data protection tools.

Efficient emergency management starts with timely identification of an emergency through the analysis of what has occurred in a target scenario and requires that all resources needed to properly handle the identified emergency could be timely accessed by authorized subjects. Internet of Things (IoT) technologies provide valid support to the development of efficient data sharing and analysis services and thus appear well suited for building emergency management applications. Data can be gathered by manifold types of smart devices which are nowadays available for different domains. For what healthcare is concerned, medical wearables and IoT technologies are enabling new forms of diagnose and care and allow the detection of emergency situations. For instance, during the COVID-19 pandemic, the OLVG Hospital in Netherlands started to experiment with wearable biosensors able to detect possible deterioration of suspected or confirmed COVID-19 patients. [1] The work conducted in Netherlands aims to improve the quality of clinical surveillance by detecting timely health risks of patients.

The management of an emergency requires granting exceptional privileges to subjects, which in an ordinary situation would not be permitted. For instance, in an ordinary situation, a physician responsible to provide treatment has to ensure that valid consent has been obtained from the patient or a delegated person before the treatment can begin. However, if an emergency occurs and the treatment is finalized to save the patient life, it can be provided without consent. Nonetheless, all granted exceptional privileges have to be immediately revoked as soon as the emergency is over.

Though a variety of access control approaches for IoT applications have been proposed in the literature, just a few of them allow regulating data sharing in an emergency situation. Almost all of these proposals rely on a permission management approach known as *break the glass(BtG)*. BtG enables users to request and then gain access to resources that would not be permitted to him/her in ordinary situations. Though BtG provides flexible emergency management, it has also drawbacks. Firstly, the accesses executed after breaking the glass should be traced for later reviews to determine whether they caused possible information leakage [75]. Secondly, the abuse of BtG policies can lead the system to an unsafe state [21]. Thus, information sharing in emergency situations can be regulated by enforcement of emergency policies that grant users all privileges needed for the management of specific emergencies as soon as they occur, instead of depending on the users' requests for exceptional access privileges.

Although not targeting the IoT domain, complementary approaches to regulate information sharing in emergency situations have been proposed in [21], [51], where emergency policies were introduced to grant subjects all privileges needed for the management of specific emergencies, as soon as they occur. Since emergency management plans are expected to be a priori defined, emergency policies could be specified in such a way to fulfill information sharing requirements elicited from the associated plan. As an example, an a priori defined protocol is expected for the above-mentioned patient monitoring scenario, which, under specific emergencies, allows medical personnel with certain functions to access patients' physiological data (e.g., in case of a severe cardiovascular issue, the privileges should be granted to cardiologists). Permission management based on emergency policies allows shorting data access time, as no request to override permission has to be issued, and data can thus be received by authorized subjects as soon as the emergency begins.

Nevertheless, none of the previous approaches target the IoT domain relying on permission management to employ emergency policies.

This void in the literature motivates us to propose an Attribute-based Access Control (ABAC) framework to regulate data sharing within MQTT-based IoT applications in ordinary and emergency situations. The proposed access control framework targets

[1] https://www.bioworld.com/articles/435384-philips-debuts-wearable-vitals-sign-patch-to-monitor-covid-19-patients-for-early-intervention

MQTT-based IoT environments since the MQTT protocol is widely adopted within IoT applications and used in various IoT scenarios. We choose ABAC for two reasons: firstly, ABAC provides outstanding flexibility, as well as for the dynamic and context-aware nature of the supported policies, these characteristics perfectly fit for the IoT environments, secondly, ABAC regulates data sharing on the basis of context properties which makes it a good fit for emergency policy support. As a matter of fact, policy selection requires to evaluate access request contexts, checking whether the subject that aims at sending and receiving an MQTT message is involved in emergency situations.

The proposed system is an extension of the framework proposed in [22] that supports fine grained access control in MQTT environments. Key novel features of the proposed framework include:

- modeling support required to: i) define the events that trigger an emergency, ii) bind events to MQTT messages, iii) specify emergency situations along with their possible evolution, and iv) specify emergency policies;
- emergency management functionalities, such as the ability to: i) detect occurrences of modeled events starting from the analysis of MQTT control packets exchanged in a monitored application, and ii) identify the possible evolution of emergency situations. For event detection, we leverage on a complex event processing(CEP) engine;
- access control capabilities, such as the ability to enforce both regular and emergency access control policies which apply to an access request issued in a specific context.

To show the feasibility of the proposed approach we apply our framework to a case study of pseudo realistic complexity related to a MQTT based health monitoring application employed in a nursing home during COVID-19 pandemic. Our framework is here employed to regulate information sharing within the considered application, with the aim to ensure that in ordinary and emergency situations data can only be accessed by authorized subjects. The proposed case study allows us to exemplify the definition of all modeling artefacts required to configure the framework for the considered application. The case study has also been employed for an early performance evaluation of the proposed approach, overall showing a reasonably low enforcement overhead.

The remainder of the chapter is organized as follows. Section 5.2 introduces a running scenario that will be used throughout the chapter to exemplify basic framework concepts, and which will be also developed into a case study. Section 5.3 presents key aspects of Complex Event Processing(CEP). Section 5.4 presents key concepts related to the proposed event modeling approach, whereas Section 5.5 introduces the foundations of our access control model. Section 5.6 provides an overview of the framework architecture, and shortly present the rationale of the enforcement mechanism, which is then more thoroughly analyzed in Section 5.7. In Section 5.8 we present a case study, and an early experimental evaluation of the framework performance.

5.2 Running Example

Let us consider an IoT application that aims at monitoring health conditions and behaviors of patients hosted in a nursing home during Covid-19 pandemic. IoT devices worn by patients and sensors deployed in the rooms where patients live allow the real-time monitoring of patients' conditions. For instance, body temperature and respiratory rate are vital signs of patients that can be acquired by wearable biosensors, whereas the locations of patients can be collected by indoor tracking bracelets. The acquired data are stored, and can therefore be visualized and analyzed by dedicated monitoring apps used by medical personnel of the nursing home, by selected relatives of the patients who can check the conditions of their kin, and even by self sufficient patients who wish to check their own conditions. Medical personnel has access to physiological and environmental data, whereas patients and relatives have limited authorizations.

Patients can occasionally face emergency conditions, which require a prompt reaction of the medical personnel. To effectively manage some emergencies, it is required to share patients data in critical conditions with external physicians with the aim to promptly identifying a proper treatment. The monitored data is also used to contrast Covid-19 diffusion. Temperature and oxygen saturation level reveal potential Covid-19 symptoms and could be used to notify physicians to make a test. The access to proximity data of infected patients, and the immediate isolation of potentially infected guests, allow contrasting Covid-19 diffusion [64].

Patients data has to be accessed by authorized users in any possible situation. The considered scenario emphasizes the need of special policies to enforce access control during emergencies.

5.3 CEP

A complex event processing (CEP) system is a framework whose primary aim is to understand what is happening in a system under analysis [39]. A CEP system is composed of a set of *event sources*, a *CEP engine*, and a group of *event sinks* [25]. Event sources are components devoted to 1) identify changes of monitored system properties, and 2) notify to the CEP engine a *primitive event* which denotes the change. A CEP engine is a tool that identifies the occurrence of specific situations in the monitored system. Situations are modeled as patterns of events, referred to as *complex events*, which occur in a time interval in the monitored system. Event sinks are notified of the occurrence of complex events by the CEP engine, and are configured to promptly react to the identified conditions. A notification is an object with fields specifying a *time annotation*, which refers to the event generation time, a *payload*, which specifies the event content and is structured as a data record, and a *type*, which constrains the structure of the *payload* [26].

Example 5.1. *Let us consider a thermometer th which is used to monitor a patient's body temperature. The events generated by this event source may have payloads composed of attributes temperature and deviceId, which respectively denote the measured temperature,* *and the device identifier. A measured temperature of 37° C at time ts can thus be represented as a JSON object: Temperature@ts: {"temperature":37, "deviceId":th}, where @ts denotes the time annotation, and {"temperature": float, "deviceId": string} is the associated type.*

Complex events are usually specified using platform specific languages. Although no universally recognized standard modeling language exists for specifying complex events, the majority of CEP engines allow specifying them within SQL-like queries [39]. In order to specify complex events abstracting from platform specific details, in this thesis, we employ the abstract event algebra presented in [39], whose operators are listed in Table 5.1.

Table 5.1: An abstract event algebra for complex event specification [39]

ce::=	pe	Primitive Event
	ce_1 ; ce_2	Sequence
	$ce_1 \vee ce_2$	Disjunction
	$ce_1 \wedge ce_2$	Conjunction
	ce^*	Iteration
	$\neg\ ce$	Negation
	$\sigma_\theta\ (ce)$	Selection
	$\pi_m(ce)$	Projection
	$[ce]_{T_1}^{T_2}$	Windowing from T_1 to T_2

A complex event *ce* is defined by composition of primitive and complex events, using a variety of operators (e.g, *sequence* (;), *disjunction* ($\vee$), *conjunction* ($\wedge$)).

Additionally, the *iteration* operator (*) allows the specification of a complex event as a set of events of the same type that occur repeatedly. In this case, *ce* occurs when the number of referred occurrences is reached.

A complex event *ce* can also be defined by *negation* ($\neg$) of another event *ce'*, meaning that *ce* only occurs if *ce'* does not occur.

Finally, *ce* can be modeled as a *selection* or *projection* of other events. The *selection* operator (σ_θ) filters events whose attribute values satisfy a condition θ, whereas the *projection* operator (π_m) extracts only part of the attributes, according to a set of mapping expressions *m*.

Any specification of a complex event *ce* can refer to events that occur in a specific time interval, specified through the *window* operator $[...]_{T_1}^{T_2}$.

Example 5.2. *Let us assume that sensors worn by patients periodically issue a primitive event RespiratoryRate*$_{pe}$*, which simply notifies the observed number of breaths per minute (bpm). A complex event Breathlessness*$_{ce}$ *which shows a shortness of breath episode observed in the last 2 days, can thus be defined as:* $(\sigma_{bpm>25}(RespiratoryRate_{pe}))_{now\ -\ 2\ days}^{now}$

5.4 Event Modeling

As presented in previous section, a CEP engine receives the primitive events as the event notifications and uses them to detect complex events. More specifically, the CEP engine accepts primitive events from the payloads and types of the event notifications and finds complex events by analyzing event notifications' contents and ordering relationships on them. In this section, we focus on an event modeling for primitive and complex events.

Let us start to focus on the modeling of primitive events, which is achieved starting from the analysis of MQTT control packets, through the specification of primitive event types. A primitive event type specifies: 1) the structure of a class of primitive events, 2) the binding criteria of the considered events to the control packets exchanged in the ecosystem, and 3) the criteria to derive the event starting from the structural characteristics of the bound control packets.

A primitive event type is therefore modelled as a tuple $\langle$*pet, adc, bcr, adf*$\rangle$, where *pet* refers to the name of the event type, *adc* is a set of pairs $\langle$*id, type*$\rangle$ that specifies the attributes that compose any event of type *pet*, *bcr* specifies the binding criteria, namely the conditions for a cp_{PB} control packet to trigger the generation of events of type *pet*, whereas *adf* is a set of pairs $\langle$*id, exp*$\rangle$, where the identifier *id* refers to an attribute declared within *adc*, whereas *exp* is an initialization expression.

Boolean expressions that specify binding criteria are defined by referring to any structural property of a candidate control packet cp_{PB} (see Table 2.1), such as, for instance, the topic, the whole payload, or a payload attribute, and employing arithmetical, set and logical operators and quantifiers, as well as predefined functions. Binding criteria specify the required characteristics of a bound control packet, referred to as *t*, referring to *t* like it was a JSON object (e.g., *t.payload* refers the payload of the control packet). The same specification criteria are also employed for the initialization expressions within component *adf*, allowing one to refer to *t*'s properties, as well as to the subject, object and environment attributes related to the publishing request context of *t*. These attributes are referred to as fields of the objects *s*, *o*, and *e*, which represent the subject, object and environment associated with the publishing request *t*.

Example 5.3. *Let us now consider the specification of a primitive event type Temp for messages published by MQTT thermometers. Let us assume that any publishing request that includes "temperature" in the topic name is bound to a primitive event of type Temp, in turn defined as a tuple* $\langle$*Temp, {"temp": float, "time": long, "pID": string}, t.TopicName.includes("temperature"), {"temp":t.Payload.temperature, "pID": o.patientID, "time": e.time}*$\rangle$. *It is worth noting that the attributes initialization is achieved referring to internal properties of the message payload,*[2] *and to object and environment attributes.*

Let us now consider the modeling of complex events. Similar to primitive events, their modeling is achieved through the specification of an event type.

A complex event type is a tuple $\langle$*cet, adc, ets, exp*$\rangle$, where *cet* specifies the name of the event type, *adc* is a set of pairs $\langle$*id, type*$\rangle$ which specify the attributes composing the payload of any event of type *pet*, *ets* is a set that includes the list of identifiers of primitive and complex event types referred to in the specification of *cet*, whereas *exp* is an expression

[2]As already mentioned in Section 2.1, we assume that message payloads are structured as JSON objects.

defined with the abstract event algebra introduced in Section 5.3, which allows initializing the value of attributes declared within *adc*. *exp* is specified by referring to the event types in *ets*, and employing the event algebra operators presented in Section 5.3).

Example 5.4. *Let us now consider the specification of complex event type Fever, used to characterize events denoting that a specified patient has had fever in the last 2 days. Fever can be defined as: ⟨Fever, {"pID": string, "temp": float}, {Temp},* $(\sigma_{temp\ >\ 38}(Temp_{pe}))^{now}_{now\ -\ 2\ days}\rangle$, *where Temp is the primitive event type introduced in Example 5.3, and* $Temp_{pe}$ *is a primitive event of type Temp.*

Similarly, let us assume that primitive event type RespiratoryRate is defined as: ⟨RespiratoryRate, {"bpm": float, "time":long, "pID": string}, t.TopicName.includes("respiratoryrate"), {"bpm": t.Payload.bpm, "pID": o.patientID, "time": e.time}⟩. Referring to Example 5.2, the complex event type Breathlessness used to represent events notifying that a patient has had shortness of breath episodes in the last 2 days could thus be specified as:
⟨Breathlessness, {"pID": string, "bpm": float}, {RespiratoryRate}, $(\sigma_{bpm\ >\ 25}(RespiratoryRate_{pe}))^{now}_{now\ -\ 2\ days}\rangle$.

The sets of primitive and complex event types specified for an application scenario are hereafter referred to as PET and CET, respectively.

5.5 Access Control Model

In this section, we present an extension of the ABAC model introduced in Section 2.2, which allows regulating data sharing within MQTT-based IoT environments in ordinary and emergency situations.

The proposed model is built on top of some key conceptual elements, respectively denoted as *emergency situation*, *emergency evolution*, and *action*, which are then used to define *emergency development plans*, *emergency scenarios* and corresponding *emergency policies.*

An *emergency situation* is a critical situation that happens suddenly and requires prompt management to avoid harmful results. An emergency can evolve into another emergency, possibly more serious or mild, or it can be solved. Any emergency is characterized by a severity level which specifies its severity. In our model, an *emergency situation* is a single stage of an emergency scenario subject to possible evolution. Therefore, we model an emergency situation *ems* as a pair ⟨*sid, lev*⟩, where *sid* specifies the emergency identifier, whereas *lev* denotes the related severity level. A severity level is an integer value in the range $[L_{min} .. L_{max}]$, configurable by the system administrator at specification time (where $L_{min}, L_{max} \in \mathbb{N}$, and $1 \leq L_{min} \leq L_{max}$).

In contrast, an *action* is a task that converts an event of complex type into a MQTT publish request packet ${cp_{PB}}^{CEP}$, and forwards it to the MQTT environment. [3] An action

[3]Actions turn MQTT clients into event sinks, which possibly could be programmed to react to the detected events.

is modeled as a tuple $\langle$*aid, cet, tp, pl*$\rangle$, where *aid* and *cet* respectively specify the identifier of the modeled action and of the referred complex event type, whereas *tp* and *pl* are expressions that allow specifying the topic and payload of $\mathrm{cp_{PB}}^{\mathrm{CEP}}$. More precisely, *tp* is an initialization expression built referring to any attribute in the payload of events of type *cet* (see Section 5.4), whereas *pl* is a set of pairs $\langle$*id, exp*$\rangle$, each specifying an attribute of the payload of $\mathrm{cp_{PB}}^{\mathrm{CEP}}$. Component *id* specifies the name of an attribute, whereas *exp* is the related attribute initialization expression.

Example 5.5. *Let us now consider the specification of action SevereBreathlessness-Notifier, which, upon detecting an event of type SevereBreathlessness denoting a serious form of shortness of breath, publishes an MQTT message which notifies the detected criticality. SevereBreathlessness can be straightforwardly specified by restricting the definition of Breathlessness in Example 5.4. SevereBreathlessnessNotifier is specified as* $\langle$*SevereBreathlessnessNotifier, SevereBreathlessness, "criticality/severebreathlessness",* {*"patientId": SevereBreathlessness*$_{ce}$*.pID, "time": SevereBreathlessness*$_{ce}$*.time, "bpm": SevereBreathlessness*$_{ce}$*.bpm*}$\rangle$*. The execution of this action causes the publishing of a message on topic criticality/severebreathlessness, with a payload characterized by fields that map those of the detected event.*

An *emergency evolution* is a transition between a pair of emergency situations, which occurs when, due to the continuous analysis of the messages exchanged in the MQTT environment operated by the CEP system, an event of complex type is detected. More formally:

Definition 5.1 (Emergency evolution)**.** *An emergency evolution ev is a tuple* $\langle$*cet, src, trg, act*$\rangle$*, where cet specifies the identifier of a complex event type in CET (see Section 5.4), src and trg respectively refer to the identifiers of the emergency situations that are left and entered when an event of type cet is detected,* [4] *whereas act refers to the identifier of an action executed when pr occurs (or* $\perp$ *if no action has to be executed).*

The occurrence of events in the monitored MQTT environment can cause: i) the starting of an emergency situation, ii) the possible evolution of an emergency situation into a more severe or modest one, or even iii) the possible resolution of an emergency situation. In order to model the possible evolution of an emergency case we hereby introduce the concept of *emergency development plan.*

Definition 5.2 (Emergency development plan)**.** *An emergency development plan edp is a tuple* $\langle$*edpi, Ev*$\rangle$*, where edpi is the identifier of the emergency development plan, whereas Ev is a set of emergency evolutions depicting the possible development of an emergency case.*

The definition of an emergency development plan *edp* has to satisfy some well-formedness rules. An evolution *ev* in the set *Ev* of *edp*, referred to as *edp.Ev*, can only refer

[4] *src* / *trg* could also refer to value $\perp$ to denote that the occurrence of an event of type *cet* activates / terminates an emergency scenario. Further details are provided in the remainder of this section, where we present the concept of emergency scenario.

as end points ⊥ or an emergency situation. Moreover, any pair of evolutions which refer to the same emergency situation within component *src*, have to specify events of different type within component *cet*. The same constraint applies to pair of evolutions which refers to ⊥ as source.

In order to intuitively represent any possible evolution of the emergency situations referred to by an emergency development plan, we represent them as state machine (stm) diagrams, where *emergency situations* are depicted as *states*, and *evolutions* as *transitions*. Each state is labeled with the identifier of the modeled emergency situation, and each transition with a complex event of a type referred to by the related evolution.

Example 5.6. *Let us consider the emergency development plan PulmonaryIssues, which is characterized by the emergency situations Dyspnea, LowOxygenSaturation, and Dyspnea-Oxygen, where Dyspnea denotes a breathing discomfort, LowOxygenSaturation a low level of oxygen-saturated hemoglobin in the blood, whereas DyspneaOxygen a combination of the previous cases. Two evolutions map the activation of the modeled emergency case, respectively entering the emergency situations Dyspnea and LowOxygenSaturation, and other two its deactivation, which exit the same emergency situations. Additional evolutions allow the transition from the emergency situation Dyspnea to DyspneaOxygen, and back, as well as from LowOxygenSaturation to DyspneaOxygen, and back. Let us assume that the above mentioned evolutions refer to events of type Breathlessness, BelowThresholdO2, Normal-BreathRate, and NormalO2Level, and, also, for the sake of simplicity, that no evolution refers to actions. The scenario is depicted by the state machine shown in Fig. 5.1.*

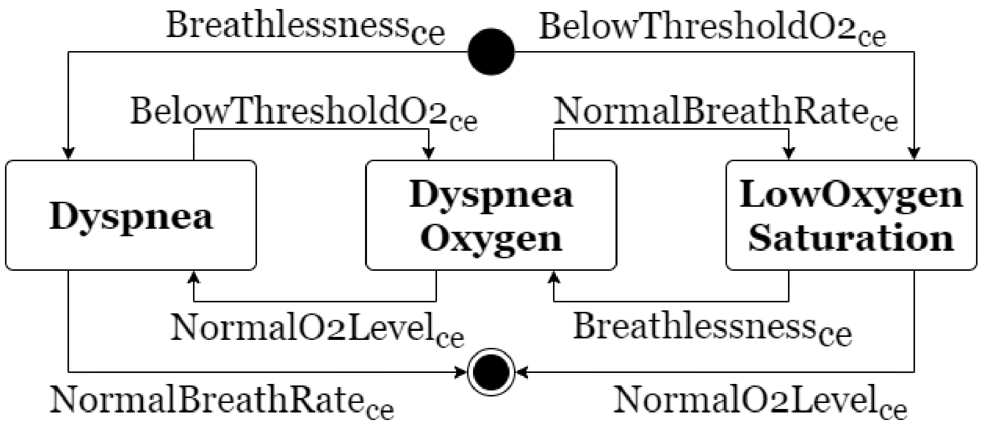

Figure 5.1: The stm diagram corresponding to the emergency development plan *PulmonaryIssues*

In contrast, the concept of *emergency scenario* is used to denote an emergency case that involves a specific set of subjects, and whose evolution is depicted by an emergency development plan.

Definition 5.3 (Emergency scenario)**.** *An emergency scenario es is a tuple ⟨esi, edp, sf⟩, where esi is the emergency scenario identifier, edp refers the identifier of the associated emergency development plan, whereas sf is a logic predicate, referred to as subject filter, which specifies under which conditions a subject is involved in es. Like parametric predicates (see Section 2.2), subject filters are defined by composition of subject attributes using mathematical and logical operators.*

At any point of the execution, an emergency scenario *es* is either inactive, or in one of the emergency situations referred to by the evolutions of the emergency development plan *edp*. More precisely, at specification time an emergency scenario *es* is *inactive*, and maintains this state until an event of type *cet* referred to by an evolution *ev* in *edp.Ev* occurs, which refers to $\perp$ as *src* component, and to an emergency situation *ems* as *trg* component. The event triggers the activation of the emergency scenario, and the entrance in the emergency situation *ems*, which is then referred to as the new current stage of *es*. Afterwards, any time an event occurs, which is referred to by an evolution *ev'* among the possible evolutions of *ems* (i.e., any evolution that refers to *ems* within component *src*), the current stage of the emergency scenario is updated. More precisely, if component *trg* of *ev'* refers to $\perp$, the emergency scenario is disabled, whereas if it refers to another emergency situation, such as, for instance, *ems'*, this new emergency is entered, and the current stage of *es* is updated to *ems'*.

It is worth noting that our model allows the specification of multiple emergency scenarios per single application, therefore a subject could be referred to by different emergency scenarios, as well as by no scenario. In addition, multiple emergency scenarios defined for an application could specify the same development plan, but different subjects.

Example 5.7. *Let us consider the definition of the emergency scenarios es_1 / es_2, respectively specifying the possible involvement of patient Bob / Mary and of the medical staff operating in the nursing home, in emergency cases modeled by the emergency development plan PulmonaryIssues (see Example 5.6). According to Def. 5.3, es_1 can be specified as ⟨es_1, PulmonaryIssues, (s.gid=patient ∧ s.uid=Bob) ∨ s.gid=medical_personnel⟩, and similarly, es_2 as ⟨es_2, PulmonaryIssues, (s.gid=patient ∧ s.uid=Mary) ∨ s.gid=medical_-personnel⟩. Although these emergency scenarios refer to the same development plan, they represent emergency cases related to two distinct patients, therefore, at any point in time the current stage of es_1 could be different from the one of es_2.*

Let us now consider the case of a subject *s* who issues an access request *ar*. If at *ar* request time no emergency scenario refers to *s*, or all emergency scenarios which refer to *s* are inactive, *s* is said to be in an *ordinary* situation, and therefore *ar* is regulated by the access control policies introduced in Section 2.2, which from now on are referred to as *ordinary policies*.

In contrast, if at *ar* request time at least one of the emergency scenarios that refer to *s* is active the request *ar* is controlled by *emergency policies*. Emergency policies regulate the ability of a subject involved in one or more emergency scenarios to send or receive MQTT messages.

Emergency policies are formally defined as follows.

Definition 5.4 (Emergency policy)**.** *An emergency policy ep is a tuple⟨s, tf, exp, pr, esf, stf⟩, where s, tf, exp, pr correspond to the homonym components in Def. 2.1, esf is an emergency scenario filter, namely an expression built referring to emergency scenario properties, resulting in a set of emergency scenarios that specify the same emergency development plan, whereas stf is a situation filter expression, which, by referring to emergency situation properties, specifies the emergency situations to which ep is applied.*

An emergency policy *ep* upon satisfaction of the parametric predicate *exp* grants to the subjects referred to by *s* the privilege to send or receive messages on topics included in *tf*, in any emergency situation denoted by *stf* of the emergency scenarios specified by *esf*.

Example 5.8. *Let us now consider the specification of an emergency policy ep which grants external specialists the access to physiological data of patients in severe conditions, with the aim to consent timely treatments. Let us assume that ep grants the access to data of patients involved in emergency scenarios that specify PulmonaryIssues as emergency development plan, and who are currently under the emergency situation DyspneaOxygen. Policy ep can be specified as: ⟨specialist, +/physiological/#, true, read, edp="PulmonaryIssues", {DyspneaOxygen}⟩. Based on Example 5.7, Mary and Bob are involved in the emergency scenarios* es_1 *and* es_2*, which specify PulmonaryIssues as emergency development plan. Therefore, according to ep a specialist can only access Bob's/Mary's data when* es_1/es_2 *specify DyspneaOxygen as current stage.*

Finally, let us shortly consider the process that allows a security administrator to specify emergency policies for a target MQTT environment.

Based on Def. 5.4, emergency policy specification can only be achieved after having defined at least one emergency scenario. In turn, at least one emergency development plan should be specified to define an emergency scenario. An emergency development plan can be defined following a step-wise process that starts with the identification of a set of emergency situations depicting possible stages of an emergency case, along with their associated severity levels. Afterwards, the security administrator needs to establish, for any considered emergency situation, if it can be entered as first stage of the emergency case, or if it can only be reached as a possible evolution of another emergency situation. Similarly, he/she needs to decide if any of the considered situations can evolve into the resolution of the emergency case. Any evolution is enabled by the occurrence of events. In order to properly label all considered evolutions, it is first required to identify the involved events, and model the related event types. Finally, the modeling of the evolutions is completed with the possible specification of actions. Action specification relies on previously mentioned modeling of complex events types, as well as on simple transformation rules that allow converting complex events into MQTT publishing requests.

Once completed the definition of emergency development plans, emergency scenarios are straightforwardly specified indicating the set of subjects potentially involved in any considered emergency case. Afterwards, the security administrator can finally focus on emergency policy specification. Emergency policies are specified as they were normal access control policies, but they make explicit reference to the emergency situations where they apply.

5.6 System Overview

Let us now focus on key architectural and behavioral properties of the proposed framework. Our proposed framework includes multiple enforcement monitors, that are used to keep a reasonably low enforcement overhead in scenarios where numerous clients are involved, and regulate the exchange of messages by MQTT clients of a monitored environment, on the basis of the specified ordinary and emergency access control policies. A NoSQL datastore is employed to keep track of metadata related to emergency management, and access control. More precisely, it stores: a) emergency scenarios along with related current stages; b) primitive and complex event types; c) ordinary and emergency policies; and d) subject, object and environment attributes employed for policy specification. A module, denoted CEP interface, is used to manage the evolution of emergency scenarios, on the basis of interactions with the monitors and a CEP engine. The possible detection of events by the CEP engine is managed by handlers embedded in the CEP interface. Any time an emergency scenario *es* is specified, two handlers are instantiated. The former is employed to manage the possible update of the referred current stage of *es*, whereas, the latter to catch CEP engine notifications denoting that no update is required. Finally, the CEP interface also embeds a MQTT publisher client responsible for issuing selected event notifications (formatted as MQTT messages) to rightful subscriber clients.

A high level view of the system architecture is shown in Figure 5.2.

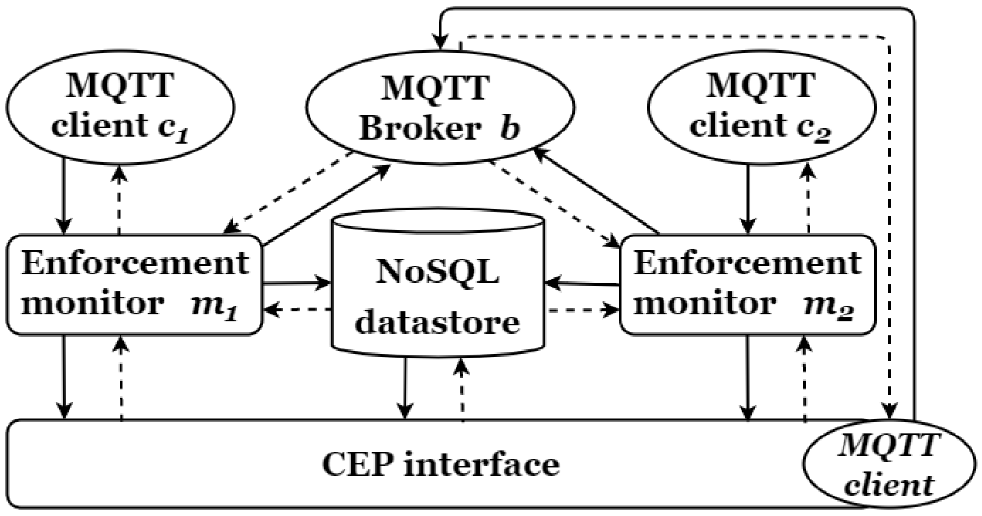

Figure 5.2: A high level view of the system architecture

In order to present the role of each component of the proposed framework, as well as the overall control flow, let us consider a simple scenario where users u_1 and u_2 respectively administer MQTT clients c_1 and c_2, which operate in a MQTT environment that hosts a broker b. Let us assume that c_1 has been configured to publish messages on topic t, whereas c_2 to subscribe the reception of messages referring to t. c_1 and c_2 have been set up to connect with broker b by means of the enforcement monitors m_1 and m_2, respectively.

In order to exchange messages, c_1 and c_2 need to connect with the MQTT broker b. Let us shortly consider the connection process of c_1 mediated by m_1 (the same process allows the connection of c_2 mediated by m_2). The process starts with a connection request issued by c_1 on behalf of u_1, denoted as ${cp_{CN}}^{c_1}$. On receipt of ${cp_{CN}}^{c_1}$, m_1: 1) opens a communication channel with b, and another one with the CEP interface, to be used to convey any communication related to c_1 requests, 2) analyzes subject attributes in the intercepted packet header deriving the identity of the requester subject, 3) forwards ${cp_{CN}}^{c_1}$ to b. The broker authenticates c_1 and sends back an acknowledgment packet ${cp_{CA}}^{c_1}$ to m_1, which in turn forwards it to c_1.

Let us now assume that, once connected, c_1 sends a publishing request ${cp_{PB}}^{c_1}$ on behalf of u_1. On receipt of ${cp_{PB}}^{c_1}$, m_1 recognizes that ${cp_{PB}}^{c_1}$ has been issued by a client, and redirects the request to the CEP interface. More precisely, it prepares a composite packet (i.e., ${cp_{PB}}^{c_1^*}$), which includes the intercepted request ${cp_{PB}}^{c_1}$, and the objects s, o, and e, with fields corresponding to the subject, object, and environment attributes associated with the request. It then issues the packet to the CEP interface, and waits for a response.

The CEP interface instantiates a control task responsible for the analysis of ${cp_{PB}}^{c_1^*}$. This task first extracts from ${cp_{PB}}^{c_1^*}$ the embedded objects, and the original request ${cp_{PB}}^{c_1}$, and identifies the requesting subject u_1 from the subject attributes. Afterwards, it checks if ${cp_{PB}}^{c_1}$ matches the binding criteria of any specified primitive event type in PET (cfr. Section 5.4). If no criterion is matched, the packet cannot trigger any emergency evolution, thus the control task notifies the monitor of the completion of the analysis. In contrast, if ${cp_{PB}}^{c_1}$ is referred to by at least one primitive event type, the task handles the generation of primitive event notifications bound to ${cp_{PB}}^{c_1}$. For any primitive event type pet in PET (see Section 5.4) which specifies binding criteria satisfied by ${cp_{PB}}^{c_1}$, the control task derives an event notification en^{pet}. The generation employs internal properties of ${cp_{PB}}^{c_1}$ and attributes extracted from ${cp_{PB}}^{c_1^*}$ referred to in the initialization expression $pet.adf$ (see Section 5.4 for more details). Let us denote with EN_{pet} the set of event notifications generated from ${cp_{PB}}^{c_1^*}$. The control task delivers EN_{pet} to the CEP engine, and waits for the completion of the analysis of this set of primitive event notifications. As soon as the control task is notified of the analysis completion by the pair of handlers associated with any emergency scenario, whose behavior is detailed in Section 5.7.2, the control task notifies the monitor that the analysis has been completed. The monitor can thus continue the processing of ${cp_{PB}}^{c_1}$.

Upon receiving the acknowledgment, the monitor selects from the NoSQL datastore the emergency scenarios that refer to u_1 as an involved subject. On the basis of the referred emergency situation of u_1 in any of the active scenarios, monitor m_1 selects from the datastore all emergency policies that apply to u_1's request ${cp_{PB}}^{c_1}$. In contrast, if there does not exist an active scenario among the selected ones, m_1 selects from the datastore all ordinary policies applicable to ${cp_{PB}}^{c_1}$. In both cases, m_1 then employs the enforcement mechanism proposed in [22] (see Section 2.2), authorizing the publishing if at least one of the selected policies grants the access.

Let us now assume that the publishing of ${cp_{PB}}^{c_1}$ has been authorized by m_1. The message is therefore issued by m_1 to the broker b, which, on the basis of the received subscriptions, forwards a copy of this packet, referred to as ${cp_{PB}}^{b}$, to c_2. The packet is then intercepted by m_2, which monitors the communication channel between c_2 and b. Since the sender of ${cp_{PB}}^{b}$ is b, m_2 derives and enforces the applicable policies without the intervention of the CEP interface. Once the identity of the receiver subject u_2 has been derived, m_2 selects from the datastore the emergency scenarios that refer to u_2 as an involved subject. In any of the selected scenarios which are not referred to as inactive, the monitor derives the current emergency situation of u_2, and then selects from the datastore all emergency policies that regulate the receiving of messages on topic t (i.e., the topic of ${cp_{PB}}^{b}$) by u_2 in any of the selected emergency situations. In contrast, if no active scenario is detected, the selection targets all ordinary policies applicable to ${cp_{PB}}^{b}$. In both cases, m_2 then proceeds applying the enforcement approach proposed in [22].

5.7 Enforcement

Let us now focus in more details on selected aspects of the proposed enforcement mechanism, instrumental to the selection of the emergency policies applicable to an access request. Selected policies are then enforced employing the mechanisms proposed in [22].

5.7.1 Event detection

A key functionality of our framework is its ability to handle the evolution of emergency scenarios, on the basis of detected complex events which trigger the entry into specific emergency situations. Reminded that an event of complex type can only occur when specific primitive events are observed, each corresponding to an MQTT client's publishing request, we hereby analyze core activities of the CEP interface instrumental for event detection, which are executed any time an MQTT client's publishing request intercepted by an enforcement monitor is forwarded to the CEP interface. More precisely, let us start to consider the *control task* instantiated by the CEP interface on receipt of a packet ${\mathrm{cp}_{\mathrm{PB}}}^{*}$ issued by an enforcement monitor. We remind that ${\mathrm{cp}_{\mathrm{PB}}}^{*}$ includes a control packet $\mathrm{cp}_{\mathrm{PB}}$ and three objects, denoted s, o and e, with fields representing the subject, object and environment attributes associated with the publishing request context (see Section 5.6).

The *control task* starts managing the generation of primitive event type notifications. This is achieved for all primitive event types belonging to PET which are bound to $\mathrm{cp}_{\mathrm{PB}}$. A primitive event type *pet* is selected iff the evaluation of the binding expression *bcr* of *pet* is satisfied by $\mathrm{cp}_{\mathrm{PB}}$.

Any selected primitive event type *spet* is then used to generate event notifications, specifying objects characterized by: 1) a *time annotation*, used for event ordering purposes, 2) a *payload*, which represents the event content, and 3) a *type*, which refers to the related event type name, implying that the structure of the event payload has to match the one specified within component *adc* of *spet*. The *time annotation* is straightforwardly derived as the timestamp related to the reception of $\mathrm{cp}_{\mathrm{PB}}{}^*$. The *type* corresponds to the value referred to by component *pet* of *spet*. Finally, the *payload* is specified by referring to the content of component *adf* of *spet*. More precisely, the control task initializes any attribute *id* referred to within component *adf* of *spet* (cfr. Section 5.4) to the value of the corresponding expression *exp*.

Example 5.9. *Let us consider the control task ct created at time rt, upon receipt of* $cp_{PB}{}^*$*. Suppose that* $cp_{PB}{}^*$ *embeds: i) a publishing request* cp_{PB} *on topic "physiological/respiratory", with a payload that includes field respiratory initialized to 27, and ii) subject, object and environment attributes indicating that* cp_{PB} *has been issued at time st by a device that monitors patient Bob conditions. Finally, let us also assume that PET includes the primitive event type RespiratoryRate introduced in Example 5.4. Since the binding expression of RespiratoryRate is satisfied by* cp_{PB}*, this primitive event type is selected for generating event notifications. The expressions in component adf of RespiratoryRate are thus evaluated for deriving the notification. As a consequence, the event notification RespiratoryRate@rt: {"bpm":27, "time":st, "patientID":"Bob"} is generated.*

Once the generation process has been completed, the control task issues the primitive event notifications to the CEP engine, and waits to be notified for the completion of the analysis of the delivered set of notifications. As soon as the pairs of handlers responsible for handling the evolution of any emergency scenario notify the completion, the control task responds to $\mathrm{cp}_{\mathrm{PB}}{}^*$ with an acknowledge message, terminating the execution.

5.7.2 Emergency management

The CEP interface manages the evolution of any specified emergency scenario *es* by means of two event handlers, denoted as *update-notifier* and *nochange-notifier* (see Section 5.6). *update-notifier* manages the detection of complex events and the possible update of *es*'s current stage, whereas, *nochange-notifier* keeps track of CEP engine analysis cycles during which no complex event is detected. These handlers are instantiated by the CEP interface at *es* specification time and then kept active as long as *es* belongs to the set of managed emergency scenarios.

Any time the CEP engine completes the analysis of a delivered set of primitive event notifications one of these handlers is invoked.

update-notifier is invoked when, on receipt of a set of primitive event notifications, a complex event *ce* of type *ecet* is detected by the CEP engine, which is referred to by an evolution *ev* of *es*. The handler is notified of the detected event, as well as of the set of primitive event notifications that have caused the event occurrence. The handler starts to select the current stage *ems* of the emergency scenario *es* from the datastore, as well as the evolution set of *es*. If *es* is not active, *ems* is initialized to $\perp$, whereas if *es* is active, it is set to the current emergency situation of *es*. If there exists an evolution *ev* whose components *src* and *cet* respectively refer to *ems* and *ecet*, *update-notifier* specifies the emergency situation referred to within component *trg* of *ev* as the new current emergency situation of *es*, propagating the update to the datastore. Let us now denote with *ct* the control task that has delivered the set of primitive event notifications which triggered the detection of *ce*. Once the update has been performed, if *ev* specifies an action *ac*, *update-notifier* gets from *ct* a copy of the composite packet $\mathrm{cp_{PB}}^*$ used for generating the notifications that caused the occurrence of *ce*. It then instantiates an *execution manager* task, which asynchronously manages the execution of *ac*, following the process detailed in Section 5.7.3, to which the derived copy is passed. Finally, *update-notifier* issues the analysis completion notification to *ct*.

In contrast, the handler *nochange-notifier* is invoked if, upon receiving a set of primitive event notifications, the CEP engine does not detect events of complex types referred to by an evolution of *es*. The handler is notified of the analyzed set of primitive event notifications, and, in turn, it notifies the analysis completion to the control task which delivered these notifications.

Example 5.10. *Let us assume that on receipt of the event notification Respiratory@rt presented in Example 5.9, the CEP engine detects an event of type Breathlessness, which is referred to by the evolutions of the emergency scenario* es_1 *(see Example 5.7), as this specifies PulmonaryIssues as emergency development plan (see Example 5.6). As a consequence, the handler update-notifier configured for* es_1 *is notified of the detected event, as well as of the primitive event notification Respiratory@rt that caused the detection. In contrast, the handler nochange-notifier is not invoked. Let us now suppose that* es_1 *is not active. The handler initializes ems to* $\perp$*, and then checks if an evolution exists, which refers to Breathlessness within component cet, and to* $\perp$ *within component src. As shown in Fig. 5.1, such an evolution exists, and specifies Dyspnea as target emergency situation. As a consequence, update-notifier activates* es_1 *specifying the emergency situation Dyspnea as the new current stage of the emergency scenario. Afterwards, since the considered evolution does not refer to an action, update-notifier notifies ct to have completed the analysis.*

5.7.3 Action execution

Let us now focus on the execution of actions associated with emergency evolutions. We hereby present the process implemented by the *execution manager* task, which is responsible to handle action executions. *Execution manager* is invoked any time the current stage of an emergency scenario *es* is updated, and component *act* (see Section 5.5) of the emergency evolution *ev* that caused the update refers to an action.

Let us assume that *execution manager* has been invoked to handle the execution of a generic action *ac*. At start time *execution manager* derives three event notifications from the copy of $\mathrm{cp_{PB}}^*$ received as input (see Section 5.7.2). The derived notifications refer to a predefined event type, respectively characterized by fields corresponding to the subjects, objects and environment attributes embedded in $\mathrm{cp_{PB}}^*$. The generated events notifications are delivered to the CEP engine, whereupon *execution manager* stays on hold waiting for a CEP engine notification. If the CEP engine notifies that no event has been detected, *execution manager* immediately terminates. In contrast, on receipt of an event *ecet*, *execution manager* generates a MQTT publishing request $\mathrm{cp_{PB}}^{\mathrm{CEP}}$ on the basis of *ecet* content. The topic of $\mathrm{cp_{PB}}^{\mathrm{CEP}}$ is initialized to the result of the evaluation of the expression referred to by component *tp* of *ac* (see Section 5.5). [5] Similarly, the payload of $\mathrm{cp_{PB}}^{\mathrm{CEP}}$ is derived iterating over the initialization expressions referred to by component *pl* of *ac*, each targeting a different payload's attribute.

Finally, $\mathrm{cp_{PB}}^{\mathrm{CEP}}$ is delivered to the broker of the monitored MQTT environment by a MQTT publisher embedded in the CEP interface (see Section 5.6) and connected to the MQTT broker at system initialization time.

Example 5.11. *Let us consider again the case introduced in Example 5.10, now assuming that the evolution which in Example 5.10 has caused the activation of es_1 refers to action SevereBreathlessnessNotifier (see Example 5.5). After the current stage of es_1 is updated, update-notifier instantiates an execution manager task emt providing as input a copy of $cp_{PB}{}^{*}$ derived from ct, and the action SevereBreathlessnessNotifier referred to by the evolution. emt first derives a primitive event notification from a built-in primitive event type that does not specify binding expressions, but simply maps as payload fields the subject, object and environment attributes in $cp_{PB}{}^{*}$. Finally, emt delivers the derived notifications to the CEP engine. On receipt of these event notifications, the CEP engine notifies the detection of an event $SevereBreathlessness_{ce}$. As a consequence, emt generates a MQTT publishing request $cp_{PB}{}^{CEP}$ on topic critical/severebreathlessness, and with a payload that maps the one of the just detected event. Finally, the generated packet is issued to the broker by the MQTT client administered by the CEP interface.*

5.8 Experimental Evaluation

In this section, we first present the application of our framework to the nursing home scenario introduced in Section 5.2, then we evaluate the framework performance with a set of experiments based on the same application scenario.

Our experiments rely on a prototype of the framework introduced in Section 5.6. Our prototype includes an enforcement monitor, defined as an extended version of the ABAC monitor for MQTT environments proposed in [22], which here has been redesigned to enforce emergency policies. Metadata related to access control and emergencies are managed by an instance of Redis,[6] a popular in memory key-value datastore. Event detection is carried out by the CEP engine Esper [7], using Event Processing Language (EPL) for implementing queries able to detect events of complex types. The CEP interface, developed in

[5]We remind that *tp* is an initialization expression built referring to any attribute in the payload of events of type *cet*

[6]https://redis.io

[7]https://www.espertech.com/esper

Java, allows managing the evolution of emergency scenarios on the basis of MQTT control packets forwarded by the enforcement monitor and events detected by the CEP engine.

5.8.1 The case study

The considered MQTT-based IoT application scenario allows detecting early symptoms of COVID-19 in nursing home patients, and tracking their close contacts. Due to the high COVID-19 mortality in extended care units [80], in such environments COVID-19 diffusion is contrasted through the preventive isolation of any identified possibly infected patient. The quarantine protocol, which is normally applied to confirmed COVID-19 cases, is here extended to any patient with early symptoms of COVID-19 who has not yet undergone a test or is still waiting for a result, and to any patient among his/her recent close contacts.

We assume that sensors worn by patients monitor physiological data, such as patients temperature, respiratory rate, and peripheral oxygen saturation, whereas patients' movements are tracked through the interaction of patients' bracelets with proximity sensors deployed in any room of the nursing home. Additional exchanged data include the prescriptions of COVID-19 tests for nursing home patients, the related results, treatment options communicated to patients, and patients consent to proceed. We assume that all devices and software modules that generate data are provided with an MQTT interface, and data are exchanged by means of the MQTT protocol. Table 5.2 exemplifies a selection of primitive event types specified for the above mentioned data, each denoting a class of primitive events derived from MQTT messages on given topics exchanged in the nursing home environment. Column *pet* specifies the identifier of the considered event type, *adc* declares all fields that compose the payload of the represented event class, *bcr* defines the conditions to be met by an MQTT message for deriving an event of the represented class, and finally, *adf* specifies the expressions that allow the initialization of payload fields.

Different groups of subjects are involved in the considered application scenario. Physicians and nurses in the medical staff of the nursing home access patients data and issue communications by means of a mobile app. Similarly, external specialists use an app to remotely check patients' conditions, and to communicate possible treatments. Patients can use an app to check their own health status. The app can also be used by registered relatives of patients subject to COVID-19 quarantine, to be updated of their kin conditions.

We model the possible evolution of a COVID-19 case as an emergency development plan characterized by the following emergency situations:

- *Suspected COVID-19*, is an emergency situation with moderate severity (level 2), related to a patient who has had COVID-19 symptoms in the last days;
- *Possible COVID-19*, is an emergency situation with mild severity (level 1) related to a patient who has been referred to as close contact of a suspected or confirmed COVID-19 patient;
- *COVID-19 asymptomatic* is an emergency situation with considerable severity (level 3), related to a confirmed COVID-19 patient with no symptom;

Table 5.2: Primitive event types specified for the case study

pet	adc	bcr	adf	description
Result	{“pid”: string, “result”: boolean, “tDate”: date, “reqId”: long, “time”: long}	t.TopicName. includes (“result”)	{“pid”:o.patientID, “result”:t.Payload.result, “tDate”:t.Payload.testDate, “reqId”:t.Payload.reqId, “time”:e.time}	Shows the results of a COVID-19 test to which a patient has undergone.
Prescription	{“pid”: string, “tDate”: date, “reqId”: long, “time”: long}	t.TopicName. includes (“prescription”)	{“pid”:o.patientID, “tDate”:t.Payload.testDate, “reqId”:t.Payload.reqId, “time”:e.time}	Shows the prescription of a COVID-19 test for a patient.
Location	{“pid”: string, “pos”: string, “time”: long}	t.TopicName. includes(“location”)	{“pid”:o.patientID, “pos”:t.Payload.location, “time”:e.time}	Shows the room where a patient is located at specified time
Temperature	{“pid”: string, “temp”: float, “time”: long}	t.TopicName. includes (“temperature”)	{“pid”:o.patientID, “temp”:t.Payload.temperature, “time”:e.time }	Shows the body temperature of a patient at a specified time.
RespiratoryRate	{“pid”: string, “bpm”: float, “time”: long}	t.TopicName. includes (“respiratory”)	{“pid”:o.patientID, “bpm”:t.Payload.respiratory, “time”:e.time}	Shows the respiratory rate of a patient at a specified time.
EstimatedSpO2	{“pid”: string, “SpO2”: float, “time”: long}	t.TopicName. includes (“saturation”)	{“pid”:o.patientID, “SpO2”:t.Payload.saturation, “time”:e.time}	Shows the peripheral oxygen saturation of a patient at a specified time.
ReqAttSet	{“cid”: string, “uid”: string, “gid”: string, “pSet”: Set(string), “relativeOf”: Set(string), “pid”: string, “ts”: long }	⊥	{“cid”: s.cid, “uid”: s.uid, “gid”: s.gid, “pSet”: s.pSet, “relativeOf”: s.relativeOf, “pid”: o.patientId, “ts”: e.time}	Maps the set of subject, object and environments attributes which characterize access requests in the considered application scenario.

Table 5.3: Complex event types specified for the case study

cet	adc	ets	exp
Symptom	{"pid": string}	{Temperature, RespiratoryRate, EstimatedSpO2}	$\pi_{\text{T.pid}}\big(\sigma_{(\text{maxT>=38} \vee \text{maxBpm>=25} \vee \text{maxSpO2<0.95}))}\big({}_{\text{T.pid}}\mathcal{G}_{\text{max(bpm) as maxBpm, max(temp) as maxT, max(SpO2) as maxSpO2}}\big(\sigma_{\text{T.pid=R.pid} \wedge \text{R.pid=S.pid}}(\text{RespiratoryRate}_{\text{pe}} \text{ as R} \wedge \text{Temperature}_{\text{pe}} \text{ as T} \wedge \text{EstimatedSpO2}_{\text{pe}} \text{ as S}) \big) \big)\big)^{\text{now}}_{\text{now - 2 days}}$
Shows any patient who has had COVID-19 symptoms in the last 2 days.			
NoSymptom	{"pid": string}	{Temperature, RespiratoryRate, EstimatedSpO2}	$\pi_{\text{T.pid}}\big(\sigma_{(\text{maxT<38} \wedge \text{maxBPM<25} \wedge \text{maxSpO2}\geq\text{95})}\big(({}_{\text{R.pid}}\mathcal{G}_{\text{max(bpm) as maxBpm, max(temp) as maxT, max(SpO2) as maxSpO2}}\big(\sigma_{\text{T.pid=R.pid} \wedge \text{R.pid=S.pid}}(\text{RespiratoryRate}_{\text{pe}} \text{ as R} \wedge \text{Temperature}_{\text{pe}} \text{ as T} \wedge \text{EstimatedSpO2}_{\text{pe}} \text{ as S}) \big) \big)\big)^{\text{now}}_{\text{now - 2 days}}$
Shows any patient who did not show Covid-19 symptoms in the last 2 days.			
SevereSymptom	{"pid": string, "bpm": float}	{RespiratoryRate}	$\pi_{\text{pid, bpm}}(\sigma_{\text{bpm>30}}\ (\text{RespiratoryRate}_{\text{pe}}))^{\text{now}}_{\text{now - 2 days}}$
Shows any patient who has had severe breathlessness episodes in the last two days, along with the observed respiratory rate.			
NoSevereSymptom	{"pid": string}	{RespiratoryRate}	$\pi_{\text{pid}}(\sigma_{\text{maxB<30}}({}_{\text{pid}}\mathcal{G}_{\text{max(bpm) as maxB}}\ (\text{RespiratoryRate}_{\text{pe}})))^{\text{now}}_{\text{now-2 days}}$
Shows any patient who did not show severe breathlessness episodes in the last two days.			
Activation	{"pid": string, "reqId": long}	{ReqAttSet}	$\pi_{\text{pid, ts as reqId}}(\text{RequestAttribute}_{\text{ce}})$
Denotes the need to isolate a patient and let him/her to undergo a COVID-19 test			

UnderTest	{"pid": string}	{LastTest, Activation, Prescription}	$\pi_{\text{L.pid}}($ $\sigma_{\text{L.pid=P.pid} \wedge \text{L.reqId=P.reqId} \wedge \text{LP.time>P.time}}($ $\text{Prescription}_{\text{ce}}$ as P $\wedge\neg\text{LastTest}_{\text{ce}}$ as L $\wedge\neg\text{Prescription}_{\text{ce}}$ as LP) $\vee\,\sigma_{\text{L.pid=A.pid} \wedge \text{A.reqId=L.reqId} \wedge \text{LA.reqId>A.reqId}}($ $\text{Activation}_{\text{ce}}$ as A $\wedge\neg\text{LastTest}_{\text{ce}}$ as L $\wedge\neg\text{Activation}_{\text{ce}}$ as LA))
Shows any patient who is waiting for the results of a test or who is going to undergo a Covid-19 test.			
LastTest	{"pid": string, testDate: date, reqId: long, result: boolean}	{Result}	$\pi_{\text{LR.pid, LR.testDate, R.reqId, R.result}}($ $\sigma_{(\text{LR.pid=R.pid})\wedge(\text{LR.tDate=R.tDate})}($ $(_{\text{pid}}\,\mathcal{G}_{\text{max(tDate) as testDate}}\,(\text{Result}_{\text{pe}}))$ as LR $\wedge\ \text{Result}_{\text{pe}}$ as R))
Specifies the results of the last Covid-19 test of a patient.			
Positive	{"pid": string}	{LastTest, UnderTest}	$\pi_{\text{L.pid}}(\sigma_{\text{L.pid=U.pid} \wedge \text{L.result}}(\neg\text{UnderTest}_{\text{ce}}$ as U $\wedge\ \text{LastTest}_{\text{ce}}$ as L))
Shows any patient whose last Covid-19 test is positive, for whom no new test has been reserved.			
Negative	{"pid": string}	{LastTest, UnderTest}	$\pi_{\text{L.pid}}(\sigma_{\text{L.pid=U.pid} \wedge\neg\ \text{L.result}}(\neg\text{UnderTest}_{\text{ce}}$ as U $\wedge\ \text{LastTest}_{\text{ce}}$ as L))
Shows any patient whose last Covid-19 test is negative, for whom no new test has been reserved.			
VisitedRoom	{"pid": string, "pos": string, "time": datetime, "date": date, "ts": long}	{Location, ReqAttSet}	$\pi_{\text{L.pid, pos, time, getDate(time) as date, ts}}($ $\sigma_{\text{R.pid=L.pid}}(\text{Location}$ as L $\wedge\ \text{ReqAttSet}_{\text{pe}}$ as R) $)_{\text{now - 10 days}}^{\text{now}}$
Shows any room visited by a patient in the last 10 days, along with the time at which he/she was in the room.			

Contact	{“pid”: string, “rpid”: string, “pos”: string, “time”: datetime, “date”: date, “ts”: long}	{Location, VisitedRoom, ReqAttSet}	$\pi_{\text{L.pid, V.pid AS rpid, V.pos, V.time, V.date, V.ts}}($ $\sigma_{(\text{V.pos=L.pos})\wedge(\text{V.time=L.time})\wedge(\text{V.pid}\neq\text{L.pid})\wedge(\text{V.pid=R.pid})\wedge(\text{V.ts=R.ts})}($ $\text{Location}_{\text{pe}}$ as L $\wedge$ $\text{VisitedRoom}_{\text{ce}}$ as V $\wedge$ $\text{ReqAttSet}_{\text{pe}}$ as R))
Shows any patient who has met patient rpid in the last 10 days, as well as the room and time at which the meeting occurred.			
CloseContact	{“pid”: string, “rpid”: string, “date”: datetime, “duration”: float, “ts”: long}	{Contact, ReqAttSet}	$\pi_{\text{C.pid, C.rpid, C.date, num\_of\_1sec\_intervals as duration, P.ts}}($ $\sigma_{\text{num\_of\_1sec\_intervals}>900}($ ${}_{\text{C.pid, C.rpid, C.date, C.ts}}\mathcal{G}_{\text{count(*) as num\_of\_1sec\_intervals}}($ $\sigma_{(\text{C.rpid=R.pid})\wedge(\text{C.ts=R.ts})}(\text{Contact}_{\text{ce}}$ as P $\wedge$ $\text{ReqAttSet}_{\text{pe}}$ as R))))
Shows any patient who, cumulatively, in a day, has stayed close to patient rpid for at least 15 minutes, along with the cumulative duration of these meetings, and the date when they occurred.			

Table 5.4: Actions involved in the COVID-19 case study

aid	cet	tp	pl
NotifyClose-Contact	CloseContact	closecontact	{"pid": CloseContact_{ce}.pid}
WarnActivation	Activation	warning	{"pid": Activation_{ce}.pid, "time": Activation_{ce}.reqId}

Table 5.5: Ordinary policies for the nursing home application

s	tf	exp	pr	description
patient	prescription	o.patientId==s.uid	r	Allows patients to be informed of COVID-19 tests they must undergo.
patient	result	o.patientId==s.uid	r	Allows patients to get the results of COVID-19 test they underwent.
patient	warning	o.patientId==s.uid	r	Allows patients to be warned of having activated a COVID-19 case.
patient	closecontact	o.patientId==s.uid	r	Allows patients to be warned of being close contacts of suspected / confirmed COVID-19 cases.
patient	treatment	o.patientId==s.uid	r	Allows patients to be informed of treatment options.
patient	consent	o.patientId==s.uid	w	Allows a patient to consent to undergo a treatment.
medical_personnel	physiological/#	o.patientId $\in$ s.pSet	r	Allows physicians to access physiological data of their patients.
medical_personnel	prescription	o.patientId $\in$ s.pSet	w	Allows physicians to prescribe COVID-19 tests for their patients.
medical_personnel	result	o.patientId $\in$ s.pSet	r	Allows physicians to receive COVID-19 test results of their patients.
medical_personnel	warning	o.patientId $\in$ s.pSet	r	Allows physicians to be notified of patients' COVID case activations.
medical_personnel	treatment	o.patientId $\in$ s.pSet	w	Allows physicians to communicate treatment options to their patients.
medical_personnel	consent	o.patientId $\in$ s.pSet	r	Allows physicians to collect the consent from their patients.
medical_personnel	bulletin	o.patientId $\in$ s.pSet	w	Allows physicians to publish medical bulletins for their patients.
medical_personnel	closecontact	o.patientId $\in$ s.pSet	r	Allows physicians to be warned of patients identified as close contact of a suspected / confirmed COVID-19 case.

- *COVID-19 symptomatic* is an emergency situation with high severity (level 4), related to a confirmed symptomatic COVID-19 patient.
- *Severe COVID-19* is an emergency situation with critical severity (level 5), related to a symptomatic COVID-19 patient with severe symptoms.

The possible evolution of a COVID-19 case is represented by the state machine in Fig-

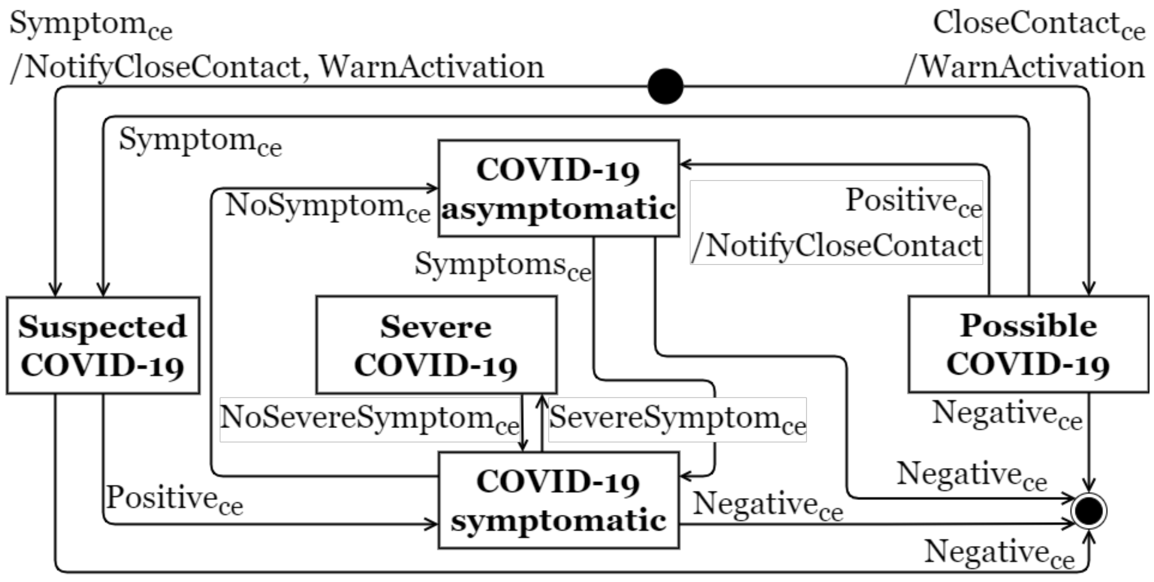

Figure 5.3: State machine representing the possible evolution of a COVID-19 case

ure 5.3, where emergency situations are represented as states, and evolutions as transitions.

Multiple emergency scenarios have then been defined (one scenario per patient), which refer *COVID-19 case* as emergency development plan, and a set of involved subjects that includes: a patient p, the medical staff of the nursing home that takes care of p, the external specialists who could be consulted, and the close relatives authorized by p to receive information about his/her health conditions.

A COVID-19 case scenario related to patient p can be activated if p shows a COVID-19 symptom or p is referred to as close contact of a suspected or confirmed COVID-19 patient. The former condition causes the entry into emergency situation *Suspected COVID-19*, whereas the latter into *Possible COVID-19*. Both emergency situations imply the need to isolate p and to let p take a COVID-19 test. If the test is negative, either emergency situations are resolved deactivating the emergency scenario, whereas, in case of a positive result with /without recent symptoms the emergency situation *COVID-19 symptomatic /COVID-19 asymptomatic* is entered. The passage from *COVID-19 symptomatic* to *COVID-19 asymptomatic* is only possible if, for some consecutive days, p does not show COVID-19 symptoms, whereas the opposite transition occurs as soon as a symptom is detected. If within *COVID-19 symptomatic* emergency situation p shows clear signs of aggravation, the emergency evolves into a *Severe COVID-19* case. The backward transition is only possible if, for some consecutive days, no severe symptom is observed. A COVID-19 case related to p is resolved in case of negative result to a COVID-19 test.

The evolution of a COVID-19 case is triggered by the occurrence of complex events. Table 5.3 exemplifies a selection of complex event types for the considered scenario, leveraging on primitive events reported in Table 5.2. Column *cet* specifies the name of the considered complex event type, *adc* specifies the set of fields composing the payload of the represented events, *ets* indicates the set of event types referred to in the specification, whereas *exp* models event specifications using the abstract event algebra introduced in Section 5.3. Event types reported in Table 5.3 consider classes of events denoting: the presence / absence of COVID-19 symptoms with various severities, the activation of a COVID-19 case, the result of the last COVID-19 test to which a patient has undergone, the rooms recently visited by a patient, and all contacts/close contacts of a patient in the last days.

The analysis of physiological and proximity data, allows deriving all patients who recently have had symptoms of COVID-19, as well as those who have had a close contact with a suspected or confirmed COVID-19 patient.

In order to promptly contrast COVID-19 diffusion, all suspected cases and their close contacts have to be immediately reported to the medical staff so that physicians could promptly isolate these patients and prescribe a test, and they can be promptly informed of their condition. This is obtained through the modeling of the actions *WarnActivation* and *NotifyCloseContact*, shown in Table 5.4. More precisely, at the early detected symptoms of COVID-19, *WarnActivation* publishes an MQTT message to inform the suspected COVID-19 patient and his/her attending physicians to be involved in an active emergency scenario. The action converts events of type *Activation* into MQTT messages, which, within the payload fields *pid* and *reqId*, simply denote the identifier of the suspected COVID-19 patient, and the timestamp at which the case has been detected. In contrast, *NotifyCloseContact* publishes an MQTT message for any detected close contact of the patient who has just entered the emergency situations *Suspected COVID-19* and *COVID-19 asymptomatic*. The action converts events of type *CloseContact* into MQTT messages. The specification of *CloseContact* (cfr. Table 5.3) shows a possible way to derive the close contacts of a patient *p*. For the proposed calculation we assume that proximity sensors check the presence of patients in any room of the nursing home at a rate of one sampling per second, and these data are then published as MQTT messages. The presence of a patient in a room is notified by primitive events of type *Location*, which also report the room (specified by field *pos*), date and time of the observation. Through the specification of complex event type *VisitedRoom*, we pick any primitive event of type *Location* observed in the last 10 days, which refers the presence of *p*. In contrast, any complex event of type *Contact* notifies that a pair of patients, which includes *p*, have been in close contact for 1 second,[8] and reports all data related to the observation. A *Contact* event is derived from a primitive event of type *Location* which notifies that, at the time specified by an event of type *VisitedRoom*, another patient was in the same room. Finally, complex events of type *CloseContact* are derived by counting all events of type *PossibleContact* which refer the same pair of patients and date. If the referred pair of patients have spent together more than 15 minutes (900 seconds) in a day, they are notified as close contacts.

Manifold privileges are granted to medical personnel operating in the nursing home, as well as to patients. A selection of the corresponding ordinary policies is reported in Table 5.5. For any policy *p*, column *s* refers the subjects who can benefit from the privileges

[8]This corresponds to the length of the interval between two consecutive samplings by proximity sensors.

granted by p, *tf* shows the topic filter expression denoting the topics of the protected messages, *exp* shows the parametric predicate that specifies under which conditions p grants the access, whereas *pr* shows the read /write privilege granted by p.

Table 5.6: Emergency policies for the COVID-19 case study

s	tf	exp	pr	esf	stf
medical_personnel	location	o.patientId ∈ s.pSet	r	edp= "COVID-19 case"	All
Allows medical personnel to check the position of their patients.					
external specialist	physiological/#	true	r	edp= "COVID-19 case"	{COVID-19 symptomatic, Severe COVID-19}
Allows external specialists to access physiological data of overt COVID-19 patients.					
relative	bulletin	o.patientId ∈ s.relativeOf	r	edp= "COVID-19 case"	All
Allows relatives to receive the medical bulletin of their kin.					
guardian	treatment	o.patientId ∈ s.guardianOf	r	edp= "COVID-19 case"	{Severe COVID-19}
Allows the guardian of a patient in severe conditions to access treatment options.					
guardian	consent	o.patientId ∈ s.relativeOf	w	edp= "COVID-19 case"	{Severe COVID-19}
Allows the guardian of a patient in severe conditions to give the consent to made his/her kin undergo specific treatments.					
guardian	result	o.patientId ∈ s.guardianOf	r	edp= "COVID-19 case"	{Severe COVID-19}
Allows the guardian of a patient in severe conditions to access his/her test results.					

According to these policies, patients can receive, through their mobile app, communications related to: i) test prescriptions and results, ii) warning messages informing them to be suspected of COVID-19 or to have been referred to as close contact of a COVID-19 case, iii) medical bulletins, and iv) treatment options. Patients can also use the app to give the consent to undergo specific treatments.

Additionally, any physician, through his/her mobile app, is allowed to: i) monitor the physiological conditions of his/her patients, ii) prescribe a COVID-19 test for his/her patients and receive the results, iii) receive notifications issued by the monitoring application, reporting the activation of new COVID-19 cases, iv) illustrate treatment options to any of his/her patient, and collect the consent to proceed with the treatment, v) issue medical bulletins to his/her patients, informing each of them about his/her conditions, and vi) access notifications issued by the monitoring application, denoting that one of his/her patients has had a close contact with a suspected or confirmed COVID-19 case.

Subjects involved in an emergency situation can benefit from additional privileges. A selection of emergency policies for the considered application scenario is reported in Table 5.6. For any emergency policy *ep*, column *esf* and *stf* specify expressions respectively denoting the set of emergency scenarios and emergency situations to which *ep* is applied, whereas columns *s*, *tf*, *exp*, and *pr* maintain the same meaning as in Table 5.5.

Under any of the considered emergency situations, physicians can access the position of their patients, as the efficient localization of suspected or possible COVID-19 patients allows their prompt isolation, and prevent the infection diffusion. To identify effective treatments for overt COVID-19 patients, or to identify patients who could require hospitalization, external specialists are also authorized to monitor physiological data of patients under the emergency situations *COVID-19 symptomatic* and *Severe COVID-19*.

In order to better bridge the gap between patients under quarantine protocol and their families, relatives are made aware of the conditions of their kin who cannot be visited during the quarantine. The access to the medical bulletin of a patient under any of the considered emergency situations is thus extended to a set of preregistered patient's relatives. Relatives can also play a fundamental role for patients in severe conditions, who, due to their health status, are unable to understand or take actions, acting as their guardians. Privileges granted to patients in ordinary situations are then applied to guardians of patients in critical conditions. For instance, in a Severe COVID-19 emergency situation, a guardian receives communication of patient's treatment options, and consents to specific treatments on his/her behalf.

5.8.2 Experiments

Let us now focus on the experiments we have carried out to evaluate the efficiency of the proposed access control approach, considering as reference scenario the case study introduced in Section 5.8.1.

For our performance evaluations we focus on the following aspects: *transmission time*, which denotes the time a published message takes for being received by a rightful subscriber, *time overhead*, which quantifies the time requested by the enforcement monitor to take a decision related to an access request, and *throughput*, namely, the average number of MQTT control packets per seconds which are analyzed by our framework.

Transmission time provides a first indication of the framework usability, as it allows quantifying the overall communication latency. However, it is a quite coarse grained property, as it shows the total duration of multiple communication phases. A client to client (*c2c*) communication in an MQTT environment is articulated into an initial client to broker (*c2b*) communication phase, during which a publishing request is issued by a client to broker, followed by a broker to client (*b2c*) phase, within which the broker forwards a copy of the received message to any rightful subscriber. Each packet issued by a client or forwarded by a broker is intercepted by the enforcement monitor, which allows the transit only if applicable policies grant it. Therefore, to assess the impact of policy enforcement on the overall transmission time, for any *c2c* communication, we keep track of the *time overhead* introduced by the enforcement monitor during the phases *c2b* and *b2c*, and, comprehensively, during the whole *c2c* communication. Similarly, *throughput* is calculated with reference to the communication phases *c2b*, *b2c*, and *c2c*.

Table 5.7: Observed performance measures

		Phase	Monitor 1	Monitor 2	Monitor 3	Avg	Tot
Target setup	Time overhead	c2b	25.25 ms	25.74 ms	25.35 ms	25.45 ms	-
		b2c	5.1 ms	5.17 ms	5.15 ms	5.14 ms	-
		c2c	30.36 ms	30.91 ms	30.50 ms	30.59 ms	-
	Transmission time	c2c	40.48 ms	44.49 ms	44.09 ms	43.02 ms	-
	Throughput	c2b	14.75 cp/s	14.69 cp/s	14.72 cp/s	14.72 cp/s	44.16 cp/s
		b2c	17.28 cp/s	17.26 cp/s	17.29 cp/s	17.28 cp/s	51.83 cp/s
		c2c	32.03 cp/s	31.94 cp/s	32.01 cp/s	32.00 cp/s	95.99 cp/s
Extreme case setup	Time overhead	c2b	62.15 ms	62.24 ms	64.03 ms	62.80 ms	-
		b2c	5.83 ms	5.73 ms	6.16 ms	5.91 ms	-
		c2c	67.98 ms	67.97 ms	70.19 ms	68.71 ms	-
	Transmission time	c2c	70.86 ms	70.77 ms	72.27 ms	71.3 ms	-
	Throughput	c2b	13.93 cp/s	13.85 cp/s	13.85 cp/s	13.87 cp/s	41.63 cp/s
		b2c	15.50 cp/s	15.46 cp/s	15.48 cp/s	15.48 cp/s	46.44 cp/s
		c2c	29.43 cp/s	29.31 cp/s	29.33 cp/s	29.36 cp/s	88.07 cp/s

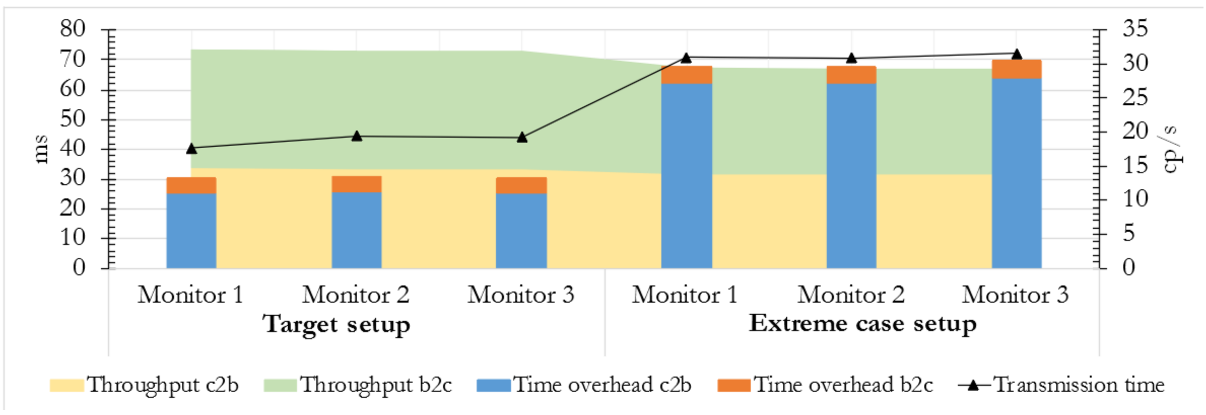

Figure 5.4: Performance analysis results

The assessment of our framework performance refers to a target setup of the monitoring application, which aims at supporting a realistic deployment tailored for a nursing home of big size. Our empirical evaluation is then complemented with a further setup, introduced to show the framework behavior in a extreme case configuration of the monitoring application.

Target setup considers a subject set of 300 patients, 60 healthcare workers among nurses and physicians, 60 relatives, and 6 specialists. In contrast, in the extreme case setup, any subject group numerousness is multiplied by 5. Subjects communication in both setups is regulated by a policy set that includes the ordinary and emergency policies presented in Section 5.8.1, and a few additional ones introduced to grant to all nursing home devices the privilege to publish sensed data.

Our experiments refer to an application deployment that includes 3 enforcement monitors, each managing the connections of one third of the subjects of each subject group. Time overhead, transmission time, and throughput are calculated at each enforcement monitor interface with the nursing home's MQTT environment. For our experiments, MQTT clients have been configured in such a way that, in the whole MQTT environment, on average, 60 publishing requests per second are generated, and the analyzed scenario refers to a period of 30 days of simulated execution of the monitoring application.

A detailed view of the computed performance measures is presented in Table 5.7, whereas Figure 5.4 shows, at aggregate level, their trend in the considered setups. In the target setup, on average, a transmission time of ~43 ms has been observed, of which, almost ~31 ms is due to the enforcement overhead. Overall, in this setup, our framework analyzes ~96 control packets per second. In contrast, the transmission time grows up to ~71 ms in the extreme case setup, with an average time overhead of ~69 ms, and a total throughput which decreases to ~88 cp/s. As visible in Figure 5.4, in each setup, the 3 enforcement monitors almost introduce the same time overhead, show similar transmission times, and comparable packet processing rates. For each monitor, time overhead related to phase *c2b* (shown in red) is significantly higher than in phase *b2c* (in blue), whereas, even though with a less marked difference, the opposite trend is observed with the throughput related to the *c2b* and *b2c* phases (respectively shown in yellow and green). This behavior is justified by the enforcement monitor activities in each communication phase. Indeed, in the *c2b* phase, on receipt of a control packet, in order to select and enforce the applicable policies, the enforcement monitor has to interact with the CEP system to check whether the intercepted packet causes the evolution of any emergency scenario, whereas, in the *b2c* phase no interaction with the CEP system is required. It is worth noting that the above mentioned differences are more accentuated in the extreme case setup, since, due to a higher number of patients to be monitored, the number of instances of COVID-19 case scenario to be managed by the CEP interface is significantly higher.

Overall, our experiments have shown satisfactory results in both setups. The observed enforcement overhead is reasonably low even in the extreme case, where the size of the monitored environment is not negligible.

Chapter 6

Conclusion and Future Work

IoT applications, which enable devices, companies, and users to join IoT ecosystems, are growing in popularity since they increase our lifestyle quality day by day. For instance, by exploiting the pervasivity of wearable technologies, manifold IoT applications assist users during their daily routines (e.g., sport training or health monitoring). However, the seamless growth of IoT enlarges the attack surface and introduces new security threats and vulnerabilities against data. Insufficient security protection mechanisms in IoT applications can cause unauthorized users to access data. To solve the security issue related to unauthorized accesses, many access control systems, which guarantee only authorized entities to access the resources, have been proposed in both academic and industrial environments. However, there are many challenges related to access control within IoT environments that still should be dealt with [68]. In this dissertation, we do a step to address two main open research challenges related to data protection in IoT environments: (i) regulating data sharing among interconnected MQTT-based IoT environments, (ii) regulating data sharing within an MQTT-based IoT environment under ordinary and emergency situations.

The access control frameworks for IoT environments proposed in the literature employ different application layer protocols(e.g., CoAP, MQTT, and XMPP [71]). Among these protocols, the access control frameworks proposed in this dissertation employ MQTT, since the MQTT protocol is widely adopted within IoT applications and used in various IoT scenarios.

The access control framework introduced in Chapter 4 regulates data sharing within interconnected IoT environments, whereas, the access control framework introduced in Chapter 5 regulates data sharing in emergency situations within an IoT environment. Both the access control frameworks proposed in this dissertation have been based on the ABAC model, since it provides outstanding flexibility and supports fine-grained, context-based access control policies [47]. These characteristics perfectly fit the IoT environments. Both proposed frameworks have been built on top of an ABAC model designed to control the communication of devices operating in an MQTT environment [22].

More specifically, in Chapter 4 we have presented an ABAC framework to control data sharing among interconnected MQTT-based IoT environments. One of the key contributions of this framework is to manage the access to data generated and exchanged within interconnected IoT environments, which distinguishes the proposed framework from the majority of approaches in the literature which control a single IoT environment. In contrast, in the proposed framework access control policies and user preferences are also employed to regulate data sharing across bridged MQTT environments. An enforcement monitor that operates as an MQTT broker proxy alters the flow of messages exchanged by the brokers of the bridged environments. Lastly, a decentralized mechanism enforces access control leveraging the joint work of monitors deployed in each environment of a bridged pair, and along the bridge. The enforcement monitor, which controls data sharing among interconnected MQTT-based IoT environments, has been designed to be easily integrated into MQTT deployments and to operate with any MQTT client and broker. We experimentally assessed the efficiency of the proposed enforcement mechanism. The performed experimental analysis has shown a reasonably low enforcement overhead in different scenarios.

We plan to improve the ABAC access control framework introduced in Chapter 4 in terms of scalability and policy management. More precisely, handling a higher number of concurrently connected clients leads to a performance drop of the enforcement mechanism. To favor the adoption of our framework in large-scale scenarios, we are investigating paralleling and load balancing techniques to automatically split up the enforcement mechanism among multiple local monitors. We are also working at complementing the framework with tools for policy management, which will allows security administrators to easily add new policies, or change existing policies.

In Chapter 5, we have presented an ABAC framework to control data sharing within an MQTT-based IoT environment under ordinary and emergency situations. The framework analyzes the MQTT messages exchanged in a monitored environment leveraging on Complex Event Processing for emergency detection. Emergency and ordinary ABAC policies are employed to regulate data sharing in emergency and ordinary situations respectively. Once the framework detects an emergency situation, the applicable emergency policies are selected to grant to the involved subjects the exceptional privileges permitted in the considered situation. We assessed the feasibility of the proposed approach with a case study related to a healthcare application that monitors nursing home patients during the COVID-19 pandemic. The case study has also been used for a preliminary evaluation of the system performance, which has been achieved by measuring message transmission time and throughput. Early experimental performance evaluations show promising results and a quite acceptable policy enforcement overhead for each configuration.

We plan to extend the abovementioned ABAC access control framework in different directions. The security administrator, who has sufficient capability and authority to organize the policies, situations, evolutions, can perform administrative operations (e.g. adding new emergency policies or updating existing ordinary policies) at runtime depending on the new requirements of the existing ecosystem. We plan to develop a tool that helps security administrators to perform those administrative operations. To promptly react to situation changes before serious incidents could happen, we also plan to add a component to predict situation changes. More precisely, we plan to leverage on machine learning to analyze ongoing messages to predict situation changes before they happen. For example, in the nursing home scenario described in Section 5.2, by predicting that the patient's condition will be worsened as a result of the pre-analysis of the patient's symptoms, medical personnel can be informed before the situation change via notifications. Thanks to the notifications, medical personnel can see that vital signs are getting worse and these signs can be used to prevent a possible emergency.

The research aspects, which have been examined above, present the direct extensions for our proposed frameworks. Now let us briefly discuss one of the most promising future extensions for access control in the IoT domain, namely the adoption of the new access control paradigm based on the Zero Trust security model.

Zero trust (ZT) [70] is an emerging paradigm in the cybersecurity field, that moves defenses from static network perimeters to focus on users, assets, and resources and their related properties. Zero Trust Architectures (ZTA) do not make any assumption on the trustworthiness of the entities [15]. The trustworthiness of entities that request access to resources is evaluated with continuous monitoring process. The monitoring process must track an entity trust level not only before the access is granted, but such level must be continuously verified and, if a relevant variation happens, the access must be stopped.

One of the main reference scenarios for ZTA is represented by IoT. Since the nature of many IoT environments is heterogeneous, it is infeasible to assume that all IoT devices authorized to communicate within the IoT environments are trusted.

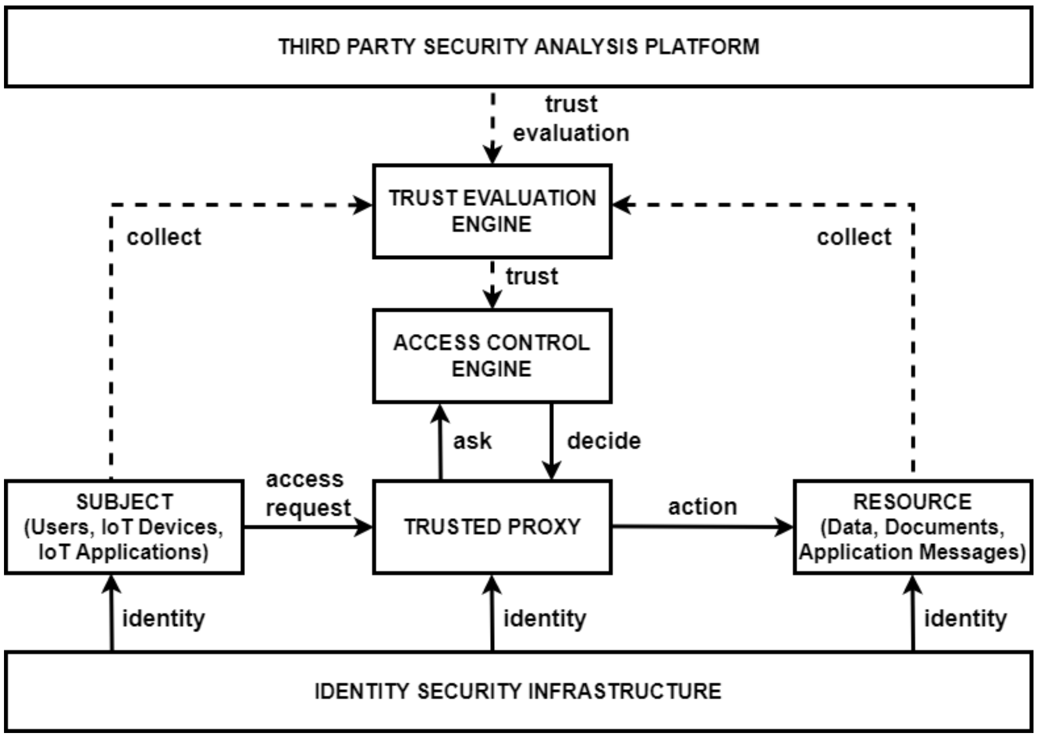
THIRD PARTY SECURITY ANALYSIS PLATFORM
trust evaluation
TRUST EVALUATION ENGINE
collect
collect
trust
ACCESS CONTROL ENGINE
ask
decide
SUBJECT (Users, IoT Devices, IoT Applications)
access request
TRUSTED PROXY
action
RESOURCE (Data, Documents, Application Messages)
identity
identity
identity
IDENTITY SECURITY INFRASTRUCTURE

Figure 6.1: Architectural Components of Zero Trust Architecture

Zero Trust security models for IoT require the adoption of strong access control mechanisms. Figure 6.1 shows the architectural components of Zero Trust Architecture that have been presented in the Gartner Report [1]. Subjects indicate users, IoT devices, or IoT applications that send access requests. Trusted Proxy intercepts access requests, communicates with the access control engine, and enforces authorization decisions. Access control engine component dynamically authorizes access requests based on subject, resource and environment attributes, in addition to the trust values sent by the trust evaluation engine. Trust evaluation engine continuously assesses trust values of subjects and analyses their access behaviours. Moreover, the trust evaluation engine keeps the access control engine informed about trust values. Third party security analysis platform is an external component that works as supplementary tools for the trust evaluation engine. Identity security infrastructure identities various entities such as subjects, trusted proxy and resources. Lastly, resources represent target objects.

The access control requirements, which are expected to be addressed to enable the adoption of ZT in the IoT domain, are as follows: *security-related metadata*, *principle of least privilege(PoLP)*, *dynamic access control decisions*, and *architectural aspects* [23].

Now let us briefly introduce these requirements.

According to ZT security models, subjects identity should be checked at any time they issue an access request, and access decisions should be based on all available *security-related metadata* such as: i) the requesting subject, ii) the target resource to be accessed, iii) the environment within which the request is issued, and iv) the security risks associated with the request. The ability of an access control framework for IoT to represent and manage properties related to subjects, objects, and context of access requests is, thus, a key instrumental feature for taking access decisions based on security risks.

Another key concept of ZT security models is the *principle of least privilege (PoLP)* [72], which essentially states that granted privileges should be the necessary minimum ones to perform a required operation. PoLP affects both access control granularity and time dimension, introducing specific requirements for the enforcement mechanisms to be used within ZT enabled IoT applications. Access control granularity should be enough fine-grained to avoid that more resources than those strictly requested could be accessed. PoLP can also be enforced by introducing temporary access privileges. Authorizations can be bound to time intervals which constrain the validity of the granted privileges. The privilege is immediately revoked once the validity period expires.

In IoT applications, the privileges should be revoked in case, before or during the access, the conditions which brought to grant the authorizations are no more satisfied. More precisely, the security risks associated with the request could be affected by any sudden change of *security-related metadata* related to an access request, which either is observed after the access is authorized and before it is executed, or during the access execution. In case the detected risk level is higher than the one derived at access request time, the access

[1]Gartner, "Zero trust architecture and solutions," Qi An Xin Group, Beijing, 2020. [Online]. Available: https://www.gartner.com/teamsiteanalytics/servePDF?g=/imagesrv/media-products/pdf/Qi-An-Xin/Qi-An-in-1-1OKONUN2.pdf

should be immediately blocked, revoking the original authorization, and possibly granting a new one with reduced privileges. Thus, the *dynamic access control decisions* should be made before and during the access.

ZT security models introduce the necessity to enforce risk aware access control policies. For any access request, applicable policies must be selected considering the behavioral patterns of the requesting subject, the sensitivity of the resources to be accessed, and current state of the system at access request time. As a direct consequence of these additional dimensions, the enforcement of fine grained security in ZT enabled IoT environments implies the need to manage and enforce a potentially huge number of access control policies [15,16]. Traditional centralized access control solutions where all decisions are taken by a single policy decision point can hardly be used in the above mentioned scenario. A key asset to cope with this complexity is therefore the adoption of decentralized and distributed AC approaches as their *architectures*, which allow improved scalability and resiliency of the enforcement mechanism.

To make the access control frameworks proposed in this thesis compliant to ZT, we plan to extend both access control frameworks with a monitoring tool that keeps track of subject behaviors. The monitoring tool checks any action performed by subjects during and after authorizations. Moreover, subject behaviors are stored in a subject behavior profile for each subject. A subject behavioral profile is employed to detect potential security risks that may affect the evaluation result of the next access request issued by its subject.

Bibliography

[1] Mohamed Abomhara and Geir M Køien. Security and privacy in the internet of things: Current status and open issues. In *2014 international conference on privacy and security in mobile systems (PRISMS)*, pages 1–8. IEEE, 2014.

[2] Alex Akinbi, Mark Forshaw, and Victoria Blinkhorn. Contact tracing apps for the covid-19 pandemic: a systematic literature review of challenges and future directions for neo-liberal societies. *Health Information Science and Systems*, 9(1):1–15, 2021.

[3] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M.D. Mickunas. A flexible, privacy-preserving authentication framework for ubiquitous computing environments. In *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pages 771–776, 2002.

[4] Asma Alshehri, James Benson, Farhan Patwa, and Ravi Sandhu. Access control model for virtual objects (shadows) communication for aws internet of things. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 175–185. ACM, 2018.

[5] Asma Alshehri and Ravi Sandhu. Access control models for cloud-enabled internet of things: A proposed architecture and research agenda. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 530–538. IEEE, 2016.

[6] Safwa Ameer, James Benson, and Ravi Sandhu. The egrbac model for smart home iot. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 457–462. IEEE, 2020.

[7] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[9] Vidyadhar Aski, Vijaypal Singh Dhaka, and Anubha Parashar. An attribute-based break-glass access control framework for medical emergencies. In *Innovations in Computational Intelligence and Computer Vision*, pages 587–595. Springer, 2021.

[10] Leonardo Babun, Kyle Denney, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. A survey on iot platforms: Communication, security, and privacy perspectives. *Computer Networks*, 192:108040, 2021.

[11] Syafril Bandara, Takeshi Yashiro, Noboru Koshizuka, and Ken Sakamura. Access control framework for api-enabled devices in smart buildings. In *2016 22nd Asia-Pacific Conference on Communications (APCC)*, pages 210–217. IEEE, 2016.

[12] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. Mqtt version 5.0. *OASIS Standard*, 2019.

[13] Sana Belguith, Sarada Prasad Gochhayat, Mauro Conti, and Giovanni Russello. Emergency access control management via attribute based encrypted qr codes. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–8. IEEE, 2018.

[14] Jorge Bernal Bernabe, Jose Luis Hernandez Ramos, and Antonio F Skarmeta Gomez. Taciot: multidimensional trust-aware access control system for the internet of things. *Soft Computing*, 20(5):1763–1779, 2016.

[15] Elisa Bertino. Zero trust architecture: Does it help? *IEEE Security Privacy*, 19(5):95–96, 2021.

[16] Elisa Bertino and Kenneth Brancik. Services for zero trust architectures-a research roadmap. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 14–20. IEEE Computer Society, 2021.

[17] Tim Bray et al. The javascript object notation (json) data interchange format. 2014.

[18] Achim D Brucker and Helmut Petritsch. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 197–206, 2009.

[19] Achim D Brucker, Helmut Petritsch, and Stefan G Weber. Attribute-based encryption with break-glass. In *IFIP International Workshop on Information Security Theory and Practices*, pages 237–244. Springer, 2010.

[20] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[21] Barbara Carminati, Elena Ferrari, and Michele Guglielmi. A system for timely and controlled information sharing in emergency situations. *IEEE Transactions on Dependable and Secure Computing*, 10(3):129–142, 2013.

[22] Pietro Colombo and Elena Ferrari. Access control enforcement within mqtt-based internet of things ecosystems. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pages 223–234, 2018.

[23] Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. Access control enforcement in iot: state of the art and open challenges in the zero trust era. In *2021 3rd IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (IEEE TPS 2021)*. IEEE, 2021.

[24] Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. Regulating data sharing across mqtt environments. *Journal of Network and Computer Applications*, 174:102907, 2021.

[25] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):1–62, 2012.

[26] Gianpaolo Cugola and Alessandro Margara. The complex event processing paradigm. In *Data Management in Pervasive Systems*, pages 113–133. Springer, 2015.

[27] Olfa Dallel, Souheil Ben Ayed, and Jamel Bel Hadj Taher. Secure iot-based emergency management system for smart buildings. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7. IEEE, 2021.

[28] Marcela T de Oliveira, Alexandros Bakas, Eugene Frimpong, Adrien ED Groot, Henk A Marquering, Antonis Michalas, and Silvia D Olabarriaga. A break-glass protocol based on ciphertext-policy attribute-based encryption to access medical records in the cloud. *Annals of Telecommunications*, pages 1–17, 2020.

[29] Theo Dimitrakos, Tezcan Dilshener, Alexander Kravtsov, Antonio La Marra, Fabio Martinelli, Athanasios Rizos, Alessandro Rosett, and Andrea Saracino. Trust aware continuous authorization for zero trust in consumer internet of things. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1801–1812. IEEE, 2020.

[30] Yuji Dong, Kaiyu Wan, Xin Huang, and Yong Yue. Contexts-states-aware access control for internet of things. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pages 666–671. IEEE, 2018.

[31] Sophie Dramé-Maigné, Maryline Laurent, and Laurent Castillo. Distributed access control solution for the iot based on multi-endorsed attributes and smart contracts. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1582–1587. IEEE, 2019.

[32] Sophie Dramé-Maigné, Maryline Laurent, Laurent Castillo, and Hervé Ganem. Centralized, distributed, and everything in between: Reviewing access control solutions for the iot. *ACM Comput. Surv.*, 54(7), September 2021.

[33] Federico Fernández, Alvaro Alonso, Lourdes Marco, and Joaquín Salvachúa. A model to enable application-scoped access control as a service for iot using oauth 2.0. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 322–324. IEEE, 2017.

[34] M. Fernandez, J. Jaimunk, and B. Thuraisingham. Privacy-preserving architecture for cloud-iot platforms. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 11–19, July 2019.

[35] Ana Ferreira, David Chadwick, Pedro Farinha, Ricardo Correia, Gansen Zao, Rui Chilro, and Luis Antunes. How to securely break into rbac: the btg-rbac model. In *2009 Annual Computer Security Applications Conference*, pages 23–31. IEEE, 2009.

[36] Ludger Fiege, Gero Mühl, and Felix C Gärtner. Modular event-based systems. *The Knowledge Engineering Review*, 17(4):359–388, 2002.

[37] Juan Carlos Fuentes Carranza and Philip WL Fong. Brokering policies and execution monitors for iot middleware. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 49–60. ACM, 2019.

[38] Alban Gabillon, Romane Gallier, and Emmanuel Bruno. Access controls for iot networks. *SN Computer Science*, 1(1):1–13, 2020.

[39] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. Complex event recognition in the big data era: a survey. *The VLDB Journal*, 29(1):313–352, 2020.

[40] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[41] Maanak Gupta, James Benson, Farhan Patwa, and Ravi Sandhu. Secure cloud assisted smart cars using dynamic groups and attribute based access control. *arXiv preprint arXiv:1908.08112*, 2019.

[42] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5-6):1189–1205, 2013.

[43] Bumjin Gwak, Jin-Hee Cho, Dongman Lee, and Heesuk Son. Taras: Trust-aware role-based access control system in public internet-of-things. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 74–85. IEEE, 2018.

[44] Dezhi Han, Yujie Zhu, Dun Li, Wei Liang, Alireza Souri, and Kuan-Ching Li. A blockchain-based auditable access control system for private data in service-centric iot environments. *IEEE Transactions on Industrial Informatics*, 2021.

[45] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012.

[46] José L Hernández-Ramos, Antonio J Jara, Leandro Marin, and Antonio F Skarmeta. Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):1–16, 2013.

[47] Vincent Hu, David Ferraiolo, D. Kuhn, A. Schnitzer, Knox Sandlin, R. Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. *National Institute of Standards and Technology Special Publication*, pages 162–800, 01 2014.

[48] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[49] Dina Hussein, Emmanuel Bertin, and Vincent Frey. A community-driven access control approach in distributed iot environments. *IEEE Communications Magazine*, 55(3):146–153, 2017.

[50] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[51] Bashar Kabbani, Romain Laborde, François Barrere, and Abdelmalek Benzekri. Specification and enforcement of dynamic authorization policies oriented by situations. In *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6. IEEE, 2014.

[52] Antonio La Marra, Fabio Martinelli, Paolo Mori, and Andrea Saracino. Implementing usage control in internet of things: a smart home use case. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 1056–1063. IEEE, 2017.

[53] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, jan 1974.

[54] Adam J Lee, Jacob T Biehl, and Conor Curry. Sensing or watching? balancing utility and privacy in sensing systems via collection and enforcement mechanisms. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pages 105–116, 2018.

[55] Han Liu, Dezhi Han, and Dun Li. Fabric-iot: A blockchain-based access control system in iot. *IEEE Access*, 8:18207–18218, 2020.

[56] Yue Liu, Qinghua Lu, Shiping Chen, Qiang Qu, Hugo O'Connor, Kim-Kwang Raymond Choo, and He Zhang. Capability-based iot access control using blockchain. *Digital Communications and Networks*, 2020.

[57] Srdjan Marinovic, Robert Craven, Jiefei Ma, and Naranker Dulay. Rumpole: a flexible break-glass access control model. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 73–82, 2011.

[58] H. A. Maw, H. Xiao, B. Christianson, and J. A. Malcolm. Btg-ac: Break-the-glass access control model for medical data in wireless sensor networks. *IEEE Journal of Biomedical and Health Informatics*, 20(3):763–774, 2016.

[59] Biswajeeban Mishra and Attila Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020.

[60] Yuta Nakamura, Yuanyu Zhang, Masahiro Sasabe, and Shoji Kasahara. Exploiting smart contracts for capability-based access control in the internet of things. *Sensors*, 20(6):1793, 2020.

[61] Fatemeh Nazerian, Homayun Motameni, and Hossein Nematzadeh. Emergency role-based access control (e-rbac) and analysis of model specifications with alloy. *Journal of information security and applications*, 45:131–142, 2019.

[62] World Health Organization et al. Online global consultation on contact tracing for covid-19, 9-11 june 2020. 2021.

[63] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112:237–262, 2017.

[64] Joseph G Ouslander and David C Grabowski. Covid-19 in nursing homes: calming the perfect storm. *Journal of the American Geriatrics Society*, 68(10):2153–2162, 2020.

[65] Jianli Pan and James McElhannon. Future edge cloud and edge computing for internet of things applications. *IEEE Internet of Things Journal*, 5(1):439–449, 2017.

[66] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM transactions on information and system security (TISSEC)*, 7(1):128–174, 2004.

[67] Jing Qiu, Zhihong Tian, Chunlai Du, Qi Zuo, Shen Su, and Binxing Fang. A survey on access control in the age of internet of things. *IEEE Internet of Things Journal*, 7(6):4682–4696, 2020.

[68] Sowmya Ravidas, Alexios Lekidis, Federica Paci, and Nicola Zannone. Access control in internet-of-things: A survey. *Journal of Network and Computer Applications*, 144:79–101, 2019.

[69] General Data Protection Regulation. Regulation eu 2016/679 of the european parliament and of the council of 27 april 2016. *Official Journal of the European Union*, 2016.

[70] Scott W Rose, Oliver Borchert, Stuart Mitchell, and Sean Connelly. Zero trust architecture. 2020.

[71] Peter Saint-Andre et al. Extensible messaging and presence protocol (xmpp): Core. 2004.

[72] Jerome H Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, 1974.

[73] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[74] Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

[75] Sigrid Schefer-Wenzl, Helena Bukvova, and Mark Strembeck. A review of delegation and break-glass models for flexible access control management. In *International conference on business information systems*, pages 93–104. Springer, 2014.

[76] Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):1–45, 2017.

[77] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[78] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[79] Qais Tasali, Christine Sublett, and Eugene Vasserman. Controlled btg: Toward flexible emergency override in interoperable medical systems. *EAI Endorsed Transactions on Security and Safety*, 6(22):e2, 2020.

[80] European Centre for Disease Prevention and Control. Increase in fatal cases of COVID-19 among long-term care facility residents in the EU/EEA and the UK. 2020.

[81] Yuanfei Tu, Jing Wang, Geng Yang, and Ben Liu. An efficient attribute-based access control system with break-glass capability for cloud-assisted industrial control system. *Mathematical Biosciences and Engineering*, 18(4):3559–3577, 2021.

[82] Dries Van Bael, Shirin Kalantari, Andreas Put, and Bart De Decker. A context-aware break glass access control system for iot environments. In *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 1–8. IEEE, 2020.

[83] Ovidiu Vermesan and Peter Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems.* River publishers, 2013.

[84] Antonio La Marra, Fabio Martinelli, Paolo Mori, Athanasios Rizos, and Andrea Saracino. Introducing usage control in mqtt. In *Computer Security*, pages 35–43. Springer, 2017.

[85] Kai Wang, Hao Yin, Wei Quan, and Geyong Min. Enabling collaborative edge computing for software defined vehicular networks. *IEEE Network*, 32(5):112–117, 2018.

[86] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the iot. *Computers*, 7(3):39, 2018.

[87] Yang Yang, Ximeng Liu, and Robert H Deng. Lightweight break-glass access control system for healthcare internet-of-things. *IEEE Transactions on Industrial Informatics*, 14(8):3610–3617, 2017.

[88] Yang Yang, Xianghan Zheng, Wenzhong Guo, Ximeng Liu, and Victor Chang. Privacy-preserving smart iot-based healthcare big data storage and self-adaptive access control system. *Information Sciences*, 479:567–592, 2019.

[89] Quan Zhang, Xiaohong Zhang, Qingyang Zhang, Weisong Shi, and Hong Zhong. Firework: Big data sharing and processing in collaborative edge environment. In *2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 20–25. IEEE, 2016.

Appendices

Appendix A

Abbreviations

Abbreviation	Meaning
ABAC	Attribute Based Access Control
ABE	Attribute Based Encryption
AC	Access Control
BtG	Break-the-Glass
CapBAC	Capability Based Access Control
CEP	Complex Event Processing
CoAP	Constrained Application Protocol
ECDSA	Elliptic Curve Digital Signature Algorithm
GDPR	General Data Protection Regulation
IoS	Internet of Sport
IoT	Internet of Things
MQTT	Message Queue Telemetry Transport
QoS	Quality of Service
PAP	Policy Administrator Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PoLP	Principle of Least Privilege
RBAC	Role Based Access Control
UCON	Usage Control
WSN	Wireless Sensor Network
ZT	Zero Trust
ZTA	Zero Trust Architecture

Appendix B

Publications

1. Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. ”Regulating data sharing across MQTT environments.” Journal of Network and Computer Applications 174 (2021): 102907.

   Abstract: Nowadays, due to the personal nature of the managed data, numerous Internet of Things (IoT) applications represent a potential threat to user privacy. In order to address this issue, several access control models have been specifically designed for IoT. The great majority of these proposals adopt centralized enforcement mechanisms designed to control the communication of IoT devices operating in the same environment. However, these approaches cannot regulate data exchange operated by devices connected to different environments. To the best of our knowledge, effective approaches capable of controlling these forms of communications are still missing. Therefore, in this paper, we do a step to fill this void, by focusing on applications built on top of MQTT, a widely used protocol for IoT. We propose an access control framework to regulate data sharing across bridged MQTT environments, on the basis of both access control policies and user preferences. The proposed approach regulates data exchange among IoT devices belonging to interconnected environments by means of a decentralized enforcement mechanism. Experimental analyses show the efficiency of the proposed approach.

2. Pietro Colombo, Elena Ferrari, and Engin Deniz Tümer. ”Access Control Enforcement in IoT: state of the art and open challenges in the Zero Trust era” in Proc. of the Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (IEEE TPS 2021), December 13 - 15, 2021.

   Abstract: Zero Trust (ZT) is a security paradigm which is nowadays finding application in different domains. One of the domain that can most benefit from ZT is represented by Internet of Things (IoT), where huge quantity of personal and sensitive data are continuously generated by IoT devices, which, in the recent years, have been the target of several security attacks. In this paper, we identify a set of access control requirements which are expected to enable the adoption of ZT in the IoT domain. Through the analysis of state of the art access control approaches currently employed in IoT ecosystems, we observe the lack of solutions capable of addressing all the identified requirements. We therefore discuss significant open challenges that still need to be addressed in this area.

3. Efficient ABAC based information sharing within MQTT environments under emergencies, uder submission.

   Abstract: Recent emergencies, such as the COVID-19 pandemic, have shown how a timely information sharing is essential to promptly and effectively react to emergencies. Internet of Things has magnified the possibility of acquiring information from different sensors and use it for emergency management and response. However, it has also amplified the possibility of information misuse and unauthorized access to information by untrusted users. Therefore, in this paper we propose an access control framework tailored to MQTT-based IoT ecosystems, which, by leveraging on Complex Event Processing is able to enforce a controlled and timely data sharing in both emergency and ordinary situations. The system has been tested with a case study that targets patient monitoring during the COVID-19 pandemic, showing promising results.