# A Bi-directional Attribute Synchronization Mechanism for Access Control in IoT Environments

Bruno Cremonezi[1], Luciano F. da Rocha[2], Alex B. Vieira[2], José Nacif[3], André L. de Oliveira[2], and Edelberto Franco Silva[2](✉)

[1] Federal University of Paraná - UFPR, Curitiba, Brazil
[2] Federal University of Juiz de Fora University - UFJF, Juiz de Fora, MG, Brazil
edelberto@ice.ufjf.br
[3] Federal University of Viçosa - UFV, Viçosa, Brazil
http://edelbertofranco.ice.ufjf.br

**Abstract.** The Attribute-Based Access Control (ABAC) model is widely used for IoT due to its capacity to express access policies through attributes, making this method granular and flexible. However, if we assume that attributes are essentially mutable, the irreducible network latency and the architectures proposed to acquire a better communication performance of the IoT expose the point where those policies are evaluated as outdated attributes. Therefore, access policies can be wrongly evaluated, resulting in consistency and security problems. In this paper, we propose a method to reduce this exposure through a bi-directional attribute synchronization capable of mapping all attributes and evaluating their current consistency after a change. If the modified attribute does not affect the access, it will remain valid. Otherwise, a revocation occurs, reducing the risks of unintended accesses. Our modeling allows demonstrating the correctness of our method and its capability to revoke every unintended access that may occur after an attribute change.

**Keywords:** IoT · Access Control · ABAC · Age of Information · UPPAAL

## 1 Introduction

The Internet of Things (IoT) is a technological trend in which common everyday objects are now equipped with sensing and communication capabilities. Therefore, the so-called IoT devices are becoming increasingly popular in our lives and today form a hyper-connected ecosystem of devices, enabling the emergence of several revolutionary applications on the market [9]. Although the benefits of this hyper-connected ecosystem to society with the provision of applications that enable automation, convenience, and effectiveness for everyday tasks, it also raises several concerns regarding the security and privacy of its users [18].

IoT devices are present in all sectors of our lives, collecting, accessing, and transferring information, often confidential or critical. Therefore, ensuring

J. Taheri et al. (Eds.): MobiCASE 2022, LNICST 495, pp. 75–88, 2023.
https://doi.org/10.1007/978-3-031-31891-7_6

that such information is not transferred to malicious locations and/or individuals, authentication and access control are essential and critical tasks for IoT devices [18]. Concerning access control, the attribute-based model (ABAC) is widely adopted in IoT applications due to its flexibility and expressiveness [16]. In an access control model, access decisions are taken based on identification attributes assigned to people, objects, or environments present in a hyperconnected IoT ecosystem against a previously defined access policy [6]. However, attributes and policies are mutable, and ideally, the entity that evaluates the access policy, i.e., the decision point, should have all these values in real-time. In practice, keeping all these values in real-time at the decision point is unfeasible. Because, even though the attributes and access policies can be accessed in real-time, there is an irreducible latency of the network that introduces a risk that some values arrive outdated at the decision point [12]. Moreover, in order to improve authorization performance, by avoiding sending attribute and policy query requests over the network, attributes and policies are usually stored in *caches*, whose values could be outdated when requested. Therefore, due to the irreducible latency of the network or outdated *caches*, some access decisions taken may be incorrect, leading to security and consistency problems [12].

In this paper, we investigate the problem of security and consistency in an IoT environment concerning the attributes of users and objects. As a first step to address this problem, we propose an attribute mapping model, a bi-directional synchronization model for all attributes, and a model capable of determining the consistent state of the attributes in the network to determine whether the access remains valid or must be revoked after an attribute update. Here, we used two approaches: if the access remains valid, an access update is performed, keeping its validity for the previously established time window; in case the access becomes invalid, its revocation occurs immediately. It is important to highlight although the consistency problem is extensively explored in practice, this work explores the formal modeling of this problem through timed automata. We seek to demonstrate the correctness of the proposal and its ability to deliver correct accesses at the end of a given execution.

The remainder of this work is organized as follows: Sect. 2 presents related works. Section 3 describes the bi-directional attribute synchronization method. Section 4 details the evaluation methodology. Section 5 discusses the results obtained. Finally, Sect. 6 presents conclusions and future directions.

## 2 Related Work

The security and consistency problem considered in this work can be addressed through a quick update of the attributes at the decision point. Within this scope, there is a number of studies addressing the issue of consistency in distributed systems, ranging from classic [1,5] to contemporary [11,15] methods. As highlighted by [17], many access control models are not fully compatible with the assumptions of distributed systems, being directed to more static environments or using reactive queries to attributes and policies.

Considering ABAC as an access control method, we have that it has premises of distributed environments due to its flexibility and granularity. Thus, we keep the focus on this access control model and its related work regarding consistency definition and credential updates. The previous related work closest to the studied concept is by *Lee and Winslett* (LW) [13,14]. Although the work of [17] is inspired by the previous one, a new perspective is considered since it evaluates the updating and not the revocation of a policy. On the other hand, our proposal evaluates the updating and consistency of attributes for use in ABAC applied to IoT and computational *fog*. In this paper, we propose the evolution and creation of a new research topic related to the investigation introduced by [17]. Our work is one of the first to evolve the concept of update and consistency operation for attributes in a distributed access policy scenario.

## 3 Assumptions and System Modeling

This work operates under the *eXtensible Access Control Markup Language* (XACML) standard. We chose XACML by being a consolidated authorization standard and explicitly defined for the access control model ABAC [3]. XACML offers specifications that cover all ways of using ABAC, from policy definitions to architecture, to support this model. Moreover, several related works available in the literature point to the XACML model as suitable to be used in an IoT scenario [7]. To illustrate it, Fig. 1 presents a diagram with the entities specified by the XACML model and the order of messages exchanged between them. According to the XACML model, four entities are needed to implement ABAC: *Policy Enforcement Point* (PEP), *Policy Decision Point* (PDP), *Policy Information Point* (PIP), and the *Policy Administration Point* (PAP). In this work, access policies are considered immutable. However, it is important to note that the entity responsible for updating and distributing such policies to the PDPs is PAP (0) [16].

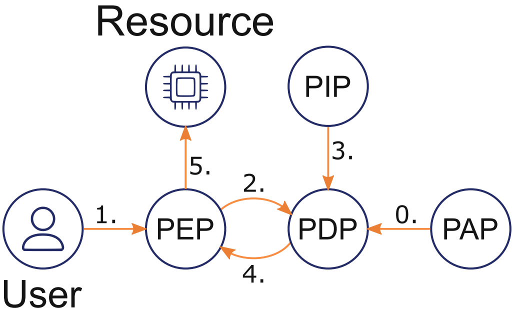

**Fig. 1.** XACML architecture.

In Fig. 1, when a given IoT user wants to access a given object, e.g., performing a reading operation on a device, the PEP intercepts this request (1) and generates an authorization request which is forwarded to the PDP (2). However, to assess whether or not this access should be authorized, the PDP looks for an access policy previously stored through the PAP and queries which attributes are needed to evaluate this policy through the PIP (3). Based on the policies and attributes, the PDP evaluates the access and sends its decision to the PEP (4), which allows or not the access of the user (5) [16].

**Users, Attributes and Policies:** In this work, for convenience, the term user is used to determine something/someone that requests access to a certain computational resource. However, it is important to note that a user could be a person, service, application, or even another IoT device. Regardless of the nature of the user, it is assumed that he/she has one or more identities with a set of attributes that describe them. Usually, in large applications - and even in medium-sized applications - it is common for the user to have several identities that represent him/her stored into several different locations [10]. The attributes of identities are mutable and vary between discrete values. Regarding access policies, this work uses immutable policies divided into rules. Each rule presents a set of attributes and the values they must have to determine if an access is valid or not. If any policy rule is satisfied, the access is valid. Otherwise, if all rules are not satisfied, the access is considered invalid [12].

**PIP, PAP, and PDP:** This work operates under an ABAC authorization environment with multiple authorities. It is assumed that there are multiple PDPs, PEPs, and PIPs distributed over a large geographic area, which serve access requests from various IoT entities. This scenario is quite common in many applications to improve the performance of the authorization process [16]. Additionally, this work determines that the PIP is segmented in a network arranged in a tree topology. The root node represents an extensive repository of attributes capable of storing the attributes of all IoT entities, and the nodes below, in turn, represent the attribute caches commonly used to obtain better authorization performance and store sub-sets of attributes. The details of the operation of this PIP located in the root are described in the next section.

**Request-Response Attribute Template:** Figure 2 illustrates the architecture proposed in this work, in which several IoT entities access each other and request authorizations from the PDPs. Note that when the PEP intercepts an access request, it is forwarded to one of the multiple PDPs, which in turn requests the attributes necessary for policy evaluation. Assuming that the PIP is segmented in a network arranged in a tree topology, the attribute request goes through multiple caches toward the root in order to find the requested attributes. If any cache has the attribute, they are sent directly through them. Otherwise, this request reaches the root of the tree that responds to attributes with a high latency [8].

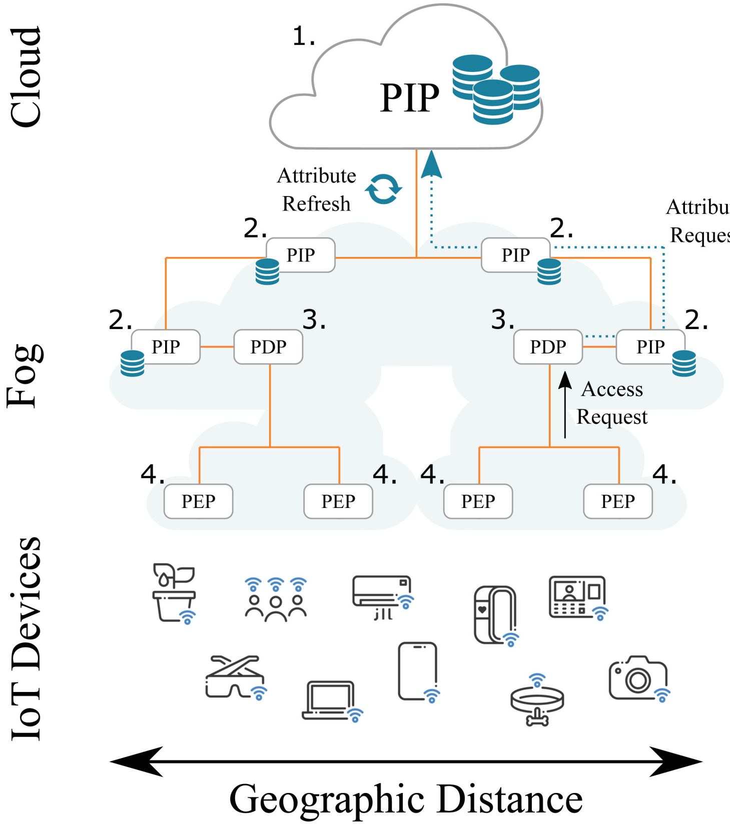

**Fig. 2.** Architecture of the authorization system

### 3.1 Problem Formulation

Managing people's attributes, IoT devices, and applications is an essential and challenging task as attributes can change frequently. For example, suppose a user accesses information from IoT devices related to multiple academic projects. This user can change position, join new projects, change location, etc. Similarly, new devices can be added to projects, other devices can be removed, and many other changes can happen. All these changes, although seemingly minor, can pose a significant challenge when using multiple PIPs to store identities. Assuming the ABAC operates in an environment with multiple authorities and several PIPs, for every access that occurs, an attribute query must be performed to determine whether the access is valid or not. However, as attributes are changeable over time and the PIP is segmented into a network arranged in a tree topology, all attributes must be updated in all caches consistently in the tree. Or, if access was previously wrongfully allowed, it must be revoked. Therefore, our main objective is to limit PDP exposure to outdated attributes, update them consistently and ensure that current and future accesses occur securely.

## 4 Bi-directional Attribute Synchronization Mechanism

In this section, the bi-directional attribute synchronization mechanism is presented. Basically, it discusses how to map and synchronize all user attributes in multiple PIPs. The purpose of our method is to allow centralized control of user attributes among several *caches* of attributes. For this, for each attribute present, the method maps its location and that of its copies, which maintains strict control over the current state of consistency of the system and, consequently, reduces the exposure of PDPs to outdated attributes. In general, even though the PIP is segmented in a network arranged in a tree topology, our method is able to offer a synchronized PIP across its entire network to allow a consistent environment of attributes. Therefore, through the proposed mechanism, it is possible to ensure that all attributes are correctly mapped and synchronized with the rest of the network.

### 4.1 Attribute Mapping

In our system model, the attributes and several copies of attributes are distributed in a PIP segmented in a network arranged in a tree topology. In order to make this PIP consistent, it is assumed that the root node of the PIP tree has a controlling role for the attributes. In this work, this entity is named as **attribute manager**. In other words, its objective, besides offering attributes, is to monitor its copies spread across the network and keep them updated. Therefore, the root contains an updated database with all attributes and provides a global and combined view of all attributes. To illustrate its function, Fig. 3 presents an example of its usefulness. Note that the left PIP (1) has the identity with the identifier "user01" and this identity has the attribute "name" with the value "Alex". Similarly, the PIP on the right (2) also has the identity with the identifier "user01", however, instead of the "name", this identity has the attribute "position" with the value "Professor". As many PIPs do not have the full view of users and attributes may be missing in others, the attribute manager allows an overview of the user. Therefore, the attribute manager has the attributes of both identities and creates a complete identity with the attributes "name" and "position". In addition, the attribute manager creates a base that points out in which PIPs the identity is stored. If this identity is removed from *cache* PIPs or added to others, these PIPs must send an attribute map update message to the manager.

### 4.2 Attribute Synchronization

As attributes are mutable, any PIP can perform an attribute update operation. However, to reflect this change in other PIPs, it is needed to perform attribute synchronization. In this work, we propose the mechanism of the bi-directional synchronization of attributes. It has this name because, at first, it is sent to the attribute manager (update occurs "up"), and the manager, from its attribute map, updates all *caches* (update occurs down). To exemplify this process, Fig. 4

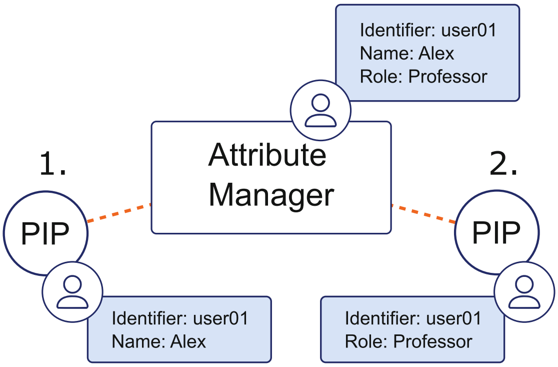

**Fig. 3.** Attribute Mapping

illustrates an attribute update. Assume that user "user01" has its identity replicated in both PIPs (1 and 2). Assume that, due to some operational change, this user's title changed from "teacher" to "researcher" and this change occurred in PIP 2. (a). PIP 2. forwards this change to the attribute manager (b), which updates the global view of the system (c). After updating the global view, it searches its attribute map in which PIPs this attribute was stored and updates them (d).

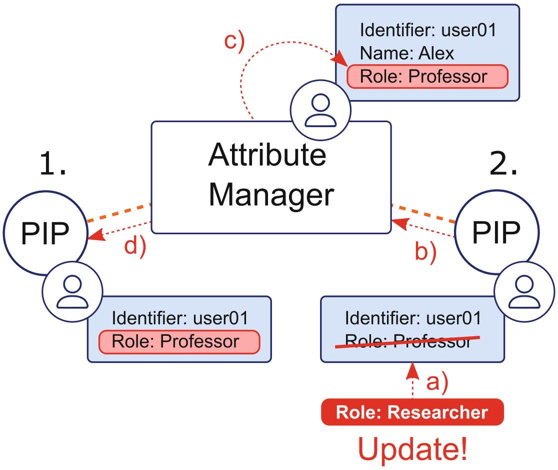

**Fig. 4.** Attribute Synchronization

It is important to note that the synchronization process takes place through an essential operation. There is a PIP, which is the source of the change (i.e., the PIP where an attribute was changed), and a PIP, which is the target of the change (PIP where the change will be propagated). Note that the map serves as a guide for this operation. Every PIP that acts as *cache* has a map that points to the attribute manager, and the manager, in turn, has a map with all the *caches* that attribute is present. Although it is not the focus of this paper, and we will not address a distributed way of managing attributes, our proposal allows PIPs *cache* to replicate attributes among themselves and map their replicas for a possible update, in case the system needs more performance and a distributed spread.

It is also worth mentioning that, in our method, it is considered that the **attribute manager** is capable of providing a correct mapping of the location of all attributes. Therefore, as long as there are no network failures or difficulties in sending messages, our method guarantees that the attributes are synchronized in the *cache* PIPs and, consequently, in the decision points for decision-making and access revalidation. However, there is no guarantee that these attributes were not modified or attacked in the *cache* PIPs. However, this scenario is considered outside the scope of this work.

### 4.3 Access Revalidation

In this work, it is assumed that when a PDP makes an access decision, it maintains a history of this permission (P), which shows which access rule was met and, consequently, which attributes were used to make the decision ($P = a1, a2, ...$). Therefore, after an attribute change, the PIP announces a change to the PDP, and the accesses are re-evaluated. In this job, permissions have three states: valid, unknown, and revoked. Therefore, instead of evaluating the entire access policy, only the permission is first evaluated.

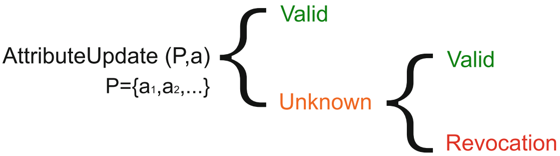

**Fig. 5.** Access Revalidation

When taking into account, the mutability of attributes, Fig. 5 illustrates how the permissions state change when the decision point receives a message that informs it that a particular attribute has been updated. If the attribute does not change permission, i.e., the modified attribute is not used in that permission, the access permission will remain **valid**. If the attribute is used, but the result of that access rule's decision does not change, the permission will also remain **valid**. If the attribute affects the access permission in such a way that the rule that granted it permission becomes invalid, this access will not become invalid, but **unknown**. For these cases, the entire access policy must be re-evaluated through the new attribute. If any other rule of the access policy is satisfied, this access permission returns to the **valid** state and is updated with the new rule that satisfies it. If the access no longer satisfies any other rule, an access **revocation** occurs, which is immediately communicated to the PEP, which suspends the user's access to the resource.

# 5 Results

For the evaluation of the proposed bi-directional synchronization mechanism of attributes, we carried out a formal verification through timed automata. Conceptually, timed automata is a generalization of finite automata to a continuous-time domain. In addition to the traditional transitions and states, the automaton also has a finite number of real variables, called clocks, whose values increase with derivative 1 concerning the passage of time. Each automaton transition can be constrained by clock values and can only occur if a particular condition is satisfied. In general, no clock modification operations exist except for the reset operation. Moreover, it is also important to note that the value of a clock can only be compared against rational constants and not against the value of other clocks [2].

## 5.1 The UPPAAL Tool

The UPPAAL [4] tool allows system modeling by defining several basic automata in an editor. In addition, the tool has a system trajectory simulator and an automatic property verification module. The verification module uses algorithms and data structures available in the literature to perform a *model-checking* of the system, which is a Cartesian product of basic automata, against properties expressed in a subset of TCTL logic (*Timed Computation Tree Logic*).

In TCTL logic, conceptually, the quantifier $A$ denotes "for every trajectory", while the quantifier $E$ denotes "there is a trajectory". For analysis, these quantifiers must be combined with the quantifiers $<>$ and $[]$, which denote, respectively, "in some state of the trajectory (*eventually*)" and "in all states of the trajectory". Therefore, the UPPAAL tool can present in its simulator an example and a counter-example of a given expression $\varphi$ when a property of type $E <> \varphi$ is true (example) or when a property of type $A[]\varphi$ is false (counter-example). For the present work, we only used the basic properties of reachability expressed by the predicate $E <> \varphi$, which denotes the existence of a trajectory in which the formula $\varphi$ becomes valid at some future time.

## 5.2 Scenario and Models

This work considers an IoT application in which a user requests access to several devices - such as thermostats, for example - in different locations [17]. Consider the scenario in Fig. 6 for illustrative purposes. Suppose Alex is a newly hired teacher and is located near PEP (1). Your identity has 3 attributes: Role, Trust, and Location. At an instant of time $t_1$, he started working, and his identity was pegged with the lowest trust level, for example, the value 1. In this application, an access policy denies users all operations on the IoT device if its trust level is 1. After some time, a top Alex user updates his trust level to the value 2 on the base PIP (1). At that moment, Alex performs access at the time instant $t_2$ and is allowed, for example, to turn the device on/off. Now suppose Alex changes his location to PEP(2). If the update level attribute has been synchronized with the PIP (2), Alex will be able to perform access at the instant $t_3$. Otherwise, the access will be denied incorrectly. This work enumerates the conditions for this access to be incorrectly denied. This situation is modeled below.

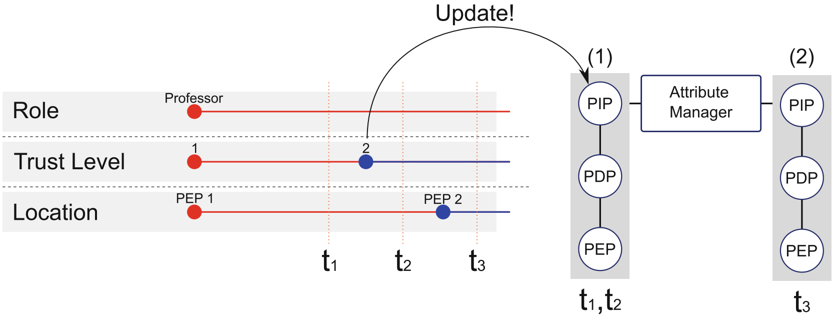

**Fig. 6.** Evaluation scenario

The automaton shown in Fig. 7 models the scenario shown in Fig. 6. This automaton controls user behavior and offers the action of requesting access (CLIENT_PLEASE_REQUEST_ACCESS), updating the attributes of an identity (PIP_PLEASE_CHANGE_ATTRIBUTES) and changing the location by the of a user (CLIENT_PLEASE_CHANGE_LOCATION). It is important to note that the "!" and "?" are synchronism between the automata. Simply put, a transition with the "?" implies a transition waiting for synchronization, whereas a transition with the operator "!" activates synchronization.

For all communication that occurs between users and XACML entities, we model sending (_SENT) and receiving messages (_RECEIVED) through an automaton that models the communication channel (Fig. 8). For every message sent (_SENT), the communication channel implies a communication delay that causes a specific time instant $t$ to pass. Only then does the other entity receive it (_RECEIVED). For example, in our channel, when an access request is sent to the PEP, the channel synchronizes this message (ACCESS_REQUEST_SENT[e]?), waits an instant of time $t$, and only then forwards this message to the PEP via the synchronization message (ACCESS_REQUEST_RECEIVED[channel]!). Note that all messages shown in Fig. 1 are modeled in this channel through ACCESS_REQUEST, DECISION_REQUEST, ATTRIBUTE_REQUEST, ACCESS_RESPONSE, DECISION_RESPONSE, ATTRIBUTE_RESPONSE synchronizations. Furthermore,

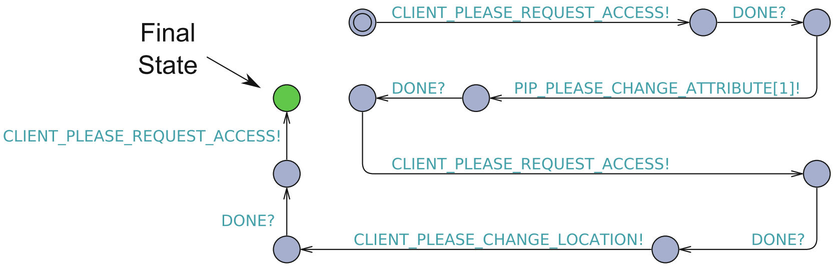
Final State
CLIENT_PLEASE_REQUEST_ACCESS!
DONE?
PIP_PLEASE_CHANGE_ATTRIBUTE[1]!
DONE?
CLIENT_PLEASE_REQUEST_ACCESS!
DONE?
CLIENT_PLEASE_CHANGE_LOCATION!
DONE?
CLIENT_PLEASE_REQUEST_ACCESS!

**Fig. 7.** Automate - Evaluation Scenario

the synchronization of updated attributes in a given PIP with the attribute manager is modeled in the GA_SYNC_ATTRIBUTE message.

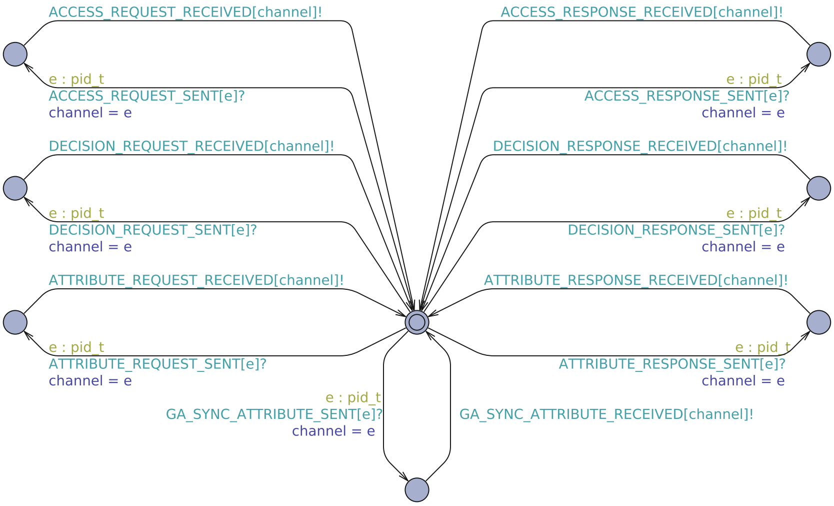
ACCESS_REQUEST_RECEIVED[channel]!
e : pid_t
ACCESS_REQUEST_SENT[e]?
channel = e
DECISION_REQUEST_RECEIVED[channel]!
e : pid_t
DECISION_REQUEST_SENT[e]?
channel = e
ATTRIBUTE_REQUEST_RECEIVED[channel]!
e : pid_t
ATTRIBUTE_REQUEST_SENT[e]?
channel = e
ACCESS_RESPONSE_RECEIVED[channel]!
e : pid_t
ACCESS_RESPONSE_SENT[e]?
channel = e
DECISION_RESPONSE_RECEIVED[channel]!
e : pid_t
DECISION_RESPONSE_SENT[e]?
channel = e
ATTRIBUTE_RESPONSE_RECEIVED[channel]!
e : pid_t
ATTRIBUTE_RESPONSE_SENT[e]?
channel = e
e : pid_t
GA_SYNC_ATTRIBUTE_SENT[e]?
channel = e
GA_SYNC_ATTRIBUTE_RECEIVED[channel]!

**Fig. 8.** Automate - Communication channel

Figure 9 illustrates the automaton that models the PEP. When intercepting a client access request (ACCESS_REQUEST_RECEIVED), the automaton sends a decision request (DECISION_REQUEST_SENT) to the PDP and waits for the PDP to respond. After responding (DECISION_RESPONSE_RECEIVED), the PEP interprets the PDP response and sends it to the client (ACCESS_RESPONSE_SENT).

Figure 10 illustrates the automaton that models the PDP. Upon receiving a decision request from the PEP (DECISION_REQUEST_RECEIVED), the automaton sends to the PIP an attribute request (ATTRIBUTE_REQUEST_SENT) and waits for the PIP to respond. After the PIP sends the requested attributes (ATTRIBUTE_RESPONSE_RECEIVED),

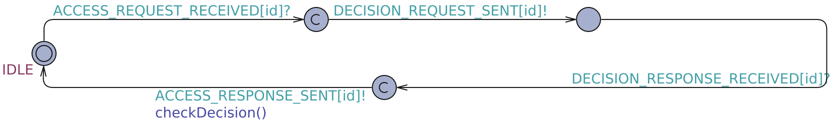
ACCESS_REQUEST_RECEIVED[id]?
C
DECISION_REQUEST_SENT[id]!
IDLE
DECISION_RESPONSE_RECEIVED[id]?
C
ACCESS_RESPONSE_SENT[id]!
checkDecision()

**Fig. 9.** Automata - PEP

the PDP evaluates the access policy and sends its decision to the PEP (DECISION_RESPONSE_SENT). It is important to note that, throughout its operation, the PDP can be requested to re-evaluate access after changing a specific attribute, updating unaffected accesses, and revoking incorrect accesses (CHECK_ACCESS_PDP) as presented in Sect. 4.3.

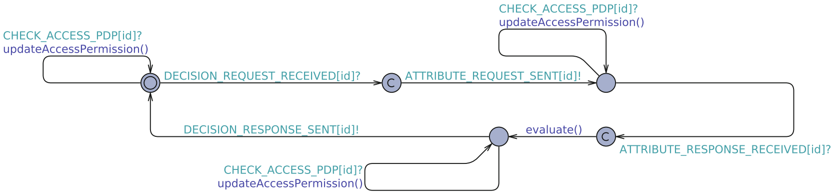

**Fig. 10.** Automate - PDP

Figure 11 illustrates the automaton that models the PIP. Its main function is to wait for an attribute request (ATTRIBUTE_REQUEST_RECEIVED) and respond appropriately (ATTRIBUTE_RESPONSE_SENT). In this work, as the attribute is mutable, its change occurs directly in the PIP (PIP_PLEASE_CHANGE_ATTRIBUTE) and triggers the sending of a message to the attribute manager to start the synchronization process. It is important to note that when the PIP receives a synchronization message from the attribute manager (SYNC_PIP), it sends the PDP a request to reassess the accesses (CHECK_ACCESS_PDP).

Finally, to close the models, Fig. 12 presents the automaton that represents the attribute manager. In general, its main function is to wait for attribute synchronization requests (GA_SYNC_ATTRIBUTE_RECEIVED) from a PIP and send the request to the other PIPs where the attribute is mapped (SYNC_PIP).

### 5.3 Formal Verification of Models

To start the formal verification of our model, we define three basic properties to be achieved. In the first moment, we want to verify if the model is free of stops and correct. Therefore, the following expression was used: $E <> deadlock$. In our tests, the time limit was 30 min, and we could not find any unexpected stops. While this does not prove that the model is free of *deadlocks*, it indicates that it is correct. Our second expression was used to check the existence of

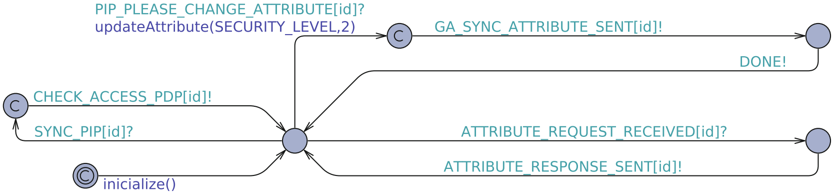
PIP_PLEASE_CHANGE_ATTRIBUTE[id]?
updateAttribute(SECURITY_LEVEL,2)
C
GA_SYNC_ATTRIBUTE_SENT[id]!
DONE!
C
CHECK_ACCESS_PDP[id]!
SYNC_PIP[id]?
ATTRIBUTE_REQUEST_RECEIVED[id]?
ATTRIBUTE_RESPONSE_SENT[id]!
C
inicialize()

**Fig. 11.** Automate - PIP

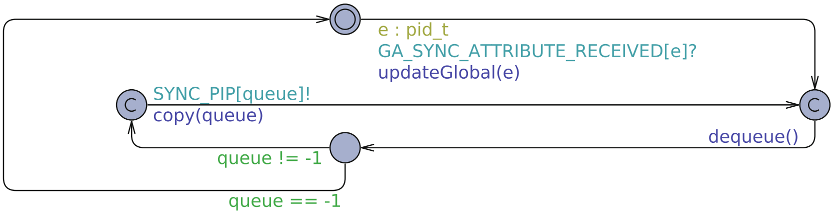

**Fig. 12.** Automate - Attribute Manager

improper accesses at the instant $t3$. For this, the following expression was used: $E <> PEP[2] \cdot invalidAuthorization()$. Translating it: “Is there a trajectory where there is invalid access in PEP 2?”. In this case, the simulator pointed it out as true and presented several examples in which if the synchronization process occurs after the access request in PEP 2, this access is granted incorrectly. Therefore, to verify that our method revokes these accesses correctly, we use the third expression: $E <> PEP[2] \cdot invalidAuthorization()\ \&\&\ SCENARIO \cdot END$. In this case, the simulator pointed out this expression as false. As much as there exist accesses that may have been granted improperly, at one point, our method is capable of re-validate them, and, at the end of the verification, there is no invalid access. Thus, we demonstrated that our method, at some point, can recognize improper access and revoke it correctly.

## 6 Conclusion

This work presented a study of real-time aspects of the bi-directional synchronization method of attributes through its modeling and verification in the UPPAAL tool. We used the Timed Automata formalism and assumed a perfect channel model to carry out a complete verification of the protocol that allowed us to identify situations and conditions for improper access. However, at the same time, our verification demonstrated that all unauthorized accesses at the end of the simulation were revoked. The results point to two directions for future work. On the one hand, it would be interesting to implement a distributed version of our method since, as much as our centralized approach removes the consistency problem, it can lead to scalability and fault tolerance problems. Moreover, we chose to use a perfect communication channel in this work. Although this demonstrated how our method works, to verify its correctness in a real environment, it is necessary to explore its behavior when messages are delayed, lost, and in other situations.

## References

1. Adya, A.: Weak consistency: a generalized theory and optimistic implementations for distributed transactions. Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and ... (1999)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
3. Anderson, A., et al.: extensible access control markup language (XACML) version 1.0. OASIS (2003)
4. Behrmann, G., et al.: UPPAAL 4.0 (2006)
5. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Comput. Surv. (CSUR) **13**(2), 185–221 (1981)
6. Bezawada, B., Haefner, K., Ray, I.: Securing home IoT environments with attribute-based access control. In: Proceedings of the Third ACM Workshop on Attribute-Based Access Control, pp. 43–53 (2018)
7. Caserio, C., Lonetti, F., Marchetti, E.: A formal validation approach for XACML 3.0 access control policy. Sensors **22**(8), 2984 (2022)
8. Cremonezi, B., Gomes Filho, A.R., Silva, E.F., Nacif, J.A.M., Vieira, A.B., Nogueira, M.: Improving the attribute retrieval on ABAC using opportunistic caches for fog-based IoT networks. Comput. Netw. **213**, 109000 (2022)
9. Dian, F.J., Vahidnia, R., Rahmati, A.: Wearables and the internet of things (IoT), applications, opportunities, and challenges: a survey. IEEE Access **8**, 69200–69211 (2020)
10. Garbis, Jason, Chapman, Jerry W..: Identity and access management. In: Garbis, J., Chapman, J.W. (eds.) Zero Trust Security, pp. 71–91. Springer, Heidelberg (2021). https://doi.org/10.1007/978-1-4842-6702-8_5
11. Harding, R., Van Aken, D., Pavlo, A., Stonebraker, M.: An evaluation of distributed concurrency control. Proc. VLDB Endow. **10**(5), 553–564 (2017)
12. Hu, V.C., et al.: Guide to attribute based access control (ABAC) definition and considerations (draft). NIST Spec. Publ. **800**(162), 1–54 (2013)
13. Lee, A.J., Winslett, M.: Safety and consistency in policy-based authorization systems. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 124–133 (2006)
14. Lee, A.J., Winslett, M.: Enforcing safety and consistency constraints in policy-based authorization systems. ACM Trans. Inf. Syst. Secur. (TISSEC) **12**(2), 1–33 (2008)
15. Perrin, M.: Distributed Systems: Concurrency and Consistency. Elsevier, Amsterdam (2017)
16. Ravidas, S., Lekidis, A., Paci, F., Zannone, N.: Access control in internet-of-things: a survey. J. Netw. Comput. Appl. **144**, 79–101 (2019)
17. Shakarami, M.: Operation and administration of access control in IoT environments. Ph.D. thesis, The University of Texas at San Antonio (2022)
18. Tawalbeh, L., Muheidat, F., Tawalbeh, M., Quwaider, M., et al.: IoT privacy and security: challenges and solutions. Appl. Sci. **10**(12), 4102 (2020)