UiO : Department of Informatics
University of Oslo

Semantic - Attribute Based Access Control

Developing an Access Control solution for IoT Gateways applied to Smart Home Care

Ugur Bayram

Master's Thesis, Spring 2020

Abstract

Access control is a security measure for restricting access to computer resources, especially in multi-user and data sharing settings. Attribute-based access control is a successor of the role-based access control that has a different approach for providing access control and provides dynamic and context-aware access control. Attribute Based Access Control (ABAC) has reached the maturity of OASIS standards with XACML 3.0 and SAML 2.0 (including profiles specific for healthcare) with existing tools like open-source Balana or PicketBox from RedHat JBoss or proprietary engines like from Axiomatics. Nevertheless, little adoption can be seen in the Health & Home care IT solutions in Europe. If in other industries the role-based access control approach can be enough, for medical data and processes the ABAC, and more granular extensions of it, are desired due to the highly sensitive and private nature of the information being accessed and the collaborative nature of the work. ABAC can handle non-trivial access policies like for collaborative access control, needed in e-Hospitals, where multiple subjects should be involved, with varying attributes and roles.

The aim of this thesis is to analyse, identify, and develop the best approach on running ABAC on a residential gateway in a smart home used by elderly people to strengthen the security of the home care.

First, we examine the ideal hardware that fulfils the basic needs such as installing a custom software as a firmware, updating/upgrading the firmware and interoperating with other wireless devices in the smart home. We examine in detail the widely-used hardware such as Raspberry PI, Residential Gateways in the market. Then, we try to implement an ABAC mechanism on a gateway device. ABAC has a distributed architecture and is usually implemented on a cloud platform. However, in this thesis we are going to implement it on a fog-node where is located inside the elderly people home (i.e., the smart home) and can be managed and maintained much easier.

Finally, we try to enrich the developed framework by adding the semantic technologies. In other words, we try to develop a Semantic Attribute Based Access Control (S-ABAC) mechanism, which makes a decision semantically and considers the semantic relationships for inferring new policies (i.e., implicit policies), for IoT environments.

Contents

List of Figures

List of Tables

Semantic - Attribute Based Access Control

Ugur Bayram

16th July 2020

Definitions

ABAC Attribute Based Access Control.

AC Access Control.

ACL Access Control List.

API Application Programming Interface.

BLE Bluetooth Low Energy.

CSI Camera Serial Interface.

DAC Discretionary Access Control.

DSI Display Serial Interface.

GDPR General Data Protection Regulation.

GPIO General Purpose Input Ouput.

HTTP Hypertext Transfer Protocol.

IBAC Identity Based Access Control.

IoT Internet of the Things.

JRE Java Runtime Environment.

MAC Mandatody Access Control.

NIST National Institute of Standards and Technology.

NPE Non-Person Entity.

OASIS Organization for the Advancement of Structured Information Standards.

OS Operating System.

OWL Web Ontology Language.

PAP Policy Administration Point.

PDP Policy Decision Point.

PEP Policy Enforcement Point.

PIP Policy Information Point.

PRP Policy Retrieval Point.

RBAC Role Based Access Control.

RDF Resource Description Framework.

REST Representational State Transfer.

RPE Request Preprocessor Extension.

RPi Raspberry PI.

S-ABAC Semantic Attribute Based Access Control.

SLA Service Level Agreement.

SPHCC Shanghai Public Health Clinical Center.

SR Semantic Reasoner.

WAR Web Application aRchive.

XACML eXtensible Access Control Markup Language.

XML eXtensible Markup Language.

CHAPTER 1

Introduction

Technological improvement continues non-stop in every field. No doubt that it brings countless benefits and let the engineers achieve certain goals. The field of Internet of the Things (IoT) receives its part from those improvements.

IoT technology aims to help in connecting different smart devices to facilitate easier operation and data sharing amongst themselves. The underneath technology collects necessary data from various smart devices, such as sensors, smartphones, and wearables. On one hand, these data are further utilized to enhance customer experience. On the other hand, due to increasing popularity of such data analytics around smart devices is expected to propel the utilization of the IoT market.

In the last two decades, the market growth has been driven by end-user industries, such as manufacturing, automotive, and healthcare thanks to the growing adoption of the IoT technology across those industries. Especially on traditional manufacturing sectors surrounded by legacy systems amid a digital transformation, the IoT becomes a game-changer for the next industrial revolution of the intelligent connectivity. This brings different approaches to industries on how to handle considerably complex systems and machines in order to increase efficiency and reduce downtime

Recent COVID-19 outbreak impacted the IoT investment and deployments all around the world. Activities at any level of organizations were taken to minimal due to the risk of virus contamination. However, with major disruptions in global healthcare and supply chains, governments, hospitals, insurers, and logistics providers were taking the duty to react as fast as possible for a more connected world that could help better address ongoing crises and mitigate future ones. In different parts of the world, smart devices played an active role in preventing the spread of the virus. Amid all the negative news, IoT technologies with their promising interconnected devices have been the silver linings this crisis has polished.

Many vendors have taken opportunity of the COVID-19 pandemic situation

by offering emerging technology-enabled solutions, especially to healthcare organizations. For instance, the Chinese healthcare organization Shanghai Public Health Clinical Center (SPHCC) used the California-based connected health startup VivaLNK's continuous temperature measuring device to monitor COVID-19 patients, which therefore reduced the risks of caregivers being exposed to the virus.

Hospitals in Vancouver, Canada installed IoT buttons called Wanda Quicktouch, which are battery operated and connect through LTE-M from Visionstate Corp that send alerts to the management of cleaning or maintenance issues that may pose risks to public safety. Facility managers can track alerts and staff response times and monitor scheduled cleaning rotations in areas of greatest footfall.

Authorities have found good use of drones to deliver medical samples and supplies to COVID-19 hotspots. The Japanese company Terra Drone brought a unique approach to serve the need. They have established the fleet of drones to transport supplies in China and claimed this has increased the speed of transport by more than 50% compared to ordinary vehicle transportation [1]. Governments and law enforcement of many countries in globe such as China, France, Spain, and the US have also used drones to monitor and ensure that citizens were obeying lockdown orders imposed due to the epidemic. Drones are also being used to spray disinfecting chemicals in some public spaces and on vehicles traveling between impacted areas.

Norwegian research center Simula collaborated with the Norwegian Institute of Public Health in order to develop a voluntary app that can reduce the time needed to track the spread of COVID-19 [2]. The applications are designed to collect location data from a user's phone. This information is used to see who an infected person has been in close contact with recently. Those persons can then be informed and take the necessary measures, both to protect themselves and to prevent further spread of infection.

The Need for Security

Although the Internet of Things (IoT) promises to bring enormous benefits, the risks associated with it cannot be ignored or underestimated. For example, wireless-enabled pacemakers and other implantable medical microchips are well-known devices carrying risks. While offering the huge benefit of remote diagnostics, at the same time, malicious access goes far beyond just a privacy breach and it is potentially life-threatening. Many low computing capability IoT devices have relatively weak security capabilities. Therefore, they are easy entry points for gatecrashers.

IoT is a non-stop growing field at a great pace. Hence, it becomes a more interesting point of attack for hackers. Globally known security firm Symantec publishes security reports every year. According to their report in 2018, the

number of attacks on IoT devices increased in only one year by 600% from 2016 to 2017, corresponding to 6000 and 50 000 reported attacks respectively. Besides the DDoS attacks, mining of cryptocurrencies has also been reported as a popular activity for hackers. Another important threat is ransomware, with WannaCry and Petya/Not Petya as the most well-known examples, resulting in a large and worldwide takedown of systems [3].

Consequently, security requires much more attention when including IoT into any business. As the number of connected IoT devices increases daily, they become like a galaxy of devices, technologies, and concepts of information. Especially when these IoT devices are implemented with poor or no security, it gets even harder to estimate and capture the consequences when malicious interventions of attackers imperil the security and privacy of IoT users and interoperability of the devices. Therefore, it is essential that the complete value chain, consists of the device, user, and network operator, integrate the required security mechanisms to guarantee end-to-end security in the communication. This will ensure gaining of trust and acceptance with the end-users, as well as, will avoid direct physical harm to humans, possibly even loss of life.

Secure communication is mainly referred to as the security features of confidentiality, the integrity of the transmitted messages, availability of services, and authentication of the sender and receiver devices. Confidentiality and integrity are well-studied security measures and might simply be accomplished through light-weight radially symmetric primitives following the accomplishment of a secret shared key. However, in order to maintain such a secret shared key, it is required to authenticate the devices and verify the identity of those devices while sending messages back and forth. Achieving authentication in a robust way is easier said than done. The authentication mechanism should be applied in each level of the value-chain, from application level to clients, and further down to device network communication. Every level contains its uniqueness which requires a characteristic solution to provide efficient authentication. For instance, in application level, multi-factor authentication plays a significant role. When it comes to user level, the biometrics is essential for an efficient authentication scheme. For the client device, encrypted and tamper-resistant physical memory, unclonable functions, and system hardening should be considered to protect the device by reducing the attack surface. Lastly, on the network level, different architectures should be considered.

Security-efficiency-cost trade-off is a major concern especially on low capacity IoT devices. While applying fine-grained security features to protect the system from intruders, one might end up compensating from efficiency. In many cases, limited computation, communication, and storage capacity are certain constraints to apply fine-grained security solutions. For example, this issue is reported in [4] as follows:

The choice of the authentication mechanism is also largely dependent on the specific use case since each use case has a different type of architecture, resulting in different requirements with respect to security features and efficiency. In any case, one of the main goals is to keep the computational, communication and storage overhead as low as possible at the side of the IoT device, which is typically the most constrained device.

1.1 Motivation

An access control system is to allow an authority to control certain access to zones and resources in a given setup. It ensures the confidentiality of the system by allowing only authorized users to access resources. It also preserves integrity by assuming that requesters are those who believed to be. Most, if not all, of recent proposals, tried to address the same problem of an access control system using a centralized approach. In such an approach, a central entity is responsible for managing complete authorization mechanisms and allows or denies requests from other entities. Yet, it is known that end-to-end security between devices and internet hosts cannot be achieved for obvious reasons. Besides, a centralized access control model may not be the ideal solution for IoT scenarios. According to [5], such an approach can bring a lack of flexibility, scalability, and usability to IoT environments where billions of devices exist and still have the potential to increase. These concerns can be avoided by a decentralized distributed approach in which 'things' are able to make authorization decisions without delegating the authorization process to an entity in a different zone.

IoT has taken an unavoidable place in human life with aim of making life easier. In many cases, they sense an environmental condition and react based on underlying resources and the purpose of using them. However, being so much in human life brings many security risks and privacy concerns. An access control system becomes a key factor behaving as a gatekeeper, securing the resources and preserving the privacy concerns. Due to the technical improvements, new access control systems aiding particular needs are being introduced. Attribute Based Access Control (ABAC) was one of them when it was first introduced on July 17, 2013, by National Institute of Standards and Technology (NIST). Since then, it became extremely popular in Information Technology world especially in field of IoT.

According to a comparative analysis of two popular access control systems, Role Based Access Control (RBAC) and ABAC, made by [6], the main drawback of ABAC is mentioned to be system complexity due to addition of attributes. While offering flexibility, dynamicity, and granularity in its functional portfolio, an ABAC system requires a good degree of storage, processor, and network

hardware infrastructure to be able to run seamlessly. This must be the reason why many organizations prefer setting up a centralized system architecture instead of decentralized. However, thank to many recent improvements, especially in hardware field; revolutionary, faster, cheaper, and smaller-in-size products enter to the market every other day. No doubt that IoT is one of the fields that benefit from these developments the most. In this thesis, we will be discussing these benefits that make it possible to implement a decentralized system architecture using ABAC, and even taking it further to improve the existing capability of ABAC by integrating with a semantic reasoner.

1.2 Research Objective

The objective of this thesis is to identify and develop the best approach of running an ABAC system on a residential gateway in order to strengthen the security of a smart home being used by an elderly person. Furthermore, enriching the capabilities of the ABAC engine will be studied by linking it to a semantic reasoner to extract the context of certain subject roles and access types. The common approach in many organizations is running such a computation-power-required access control system in the cloud area, wherein theory unlimited infrastructural resource exists. By applying this objective, a new approach will be introduced to decentralize a heavyweight access control system and move it from the cloud-area to the fog-area where it will be closer to IoT devices.

1.3 Research Questions

This section presents the research questions that this thesis project addresses. Towards to the objectives defined in Section 1.2, this thesis addresses the following research questions:

RQ1: What is the importance of the fine-grained access control for IoT settings?

RQ2: Can ABAC work for simple IoT devices on the fog-area?

RQ3: What is the ideal gateway device for this need?

RQ4: How can we link an ABAC engine to a Semantic Reasoner having a simple ontology?

1.4 Research Methodology

Gordona Dodig-Crnkovic describes research methodologies in three different scientific categories: Theoretical Computer Science, Experimental Computer Science, and Computer Simulation. He describes the last one as follows:

> **Experimental Computer Science** is most effective on problems that require complex software such as the creation of software development environments, or the construction of tools to solve constrained optimization problems. The approach is largely to identify concepts that facilitate solutions to a problem and then evaluate solutions through construction of prototype systems [7].

The research in this thesis adheres to the experimental computer science research methodology as we will experiment by proposing a solution to a real-world problem and creating a prototype application using different technologies and resources, and then evaluate the solution.

The thesis subject is based on an open-source access control system, ABAC, provided by Authzforce team. The initial task is to identify minimum hardware requirements and then decide the ideal hardware device that will host the complete prototype solution. In order to plan the software design and the implementation, it is essential to gather required knowledge on popular access control systems and more specifically on ABAC. My main sources to gather the required knowledge and to do a literature review are University of Oslo Library page (https://www.ub.uio.no/english) for articles, books and conference papers; University of Oslo Research Archive (https://www.duo.uio.no) for thesis projects that were done on the same or similar fields; and other internet sources for the technical research. Furthermore, in order to obtain insights in ABAC, my main sources are Authzforce developer guides, documentations and source-codes on the GitHub.

1.5 Outline

The thesis project is organized as follow:

- The first Chapter describes the headline of the thesis and states the problem. It is also an introduction to this master thesis containing the main motivation of selecting the thesis subject, list of the research questions that we want to achieve, and methods.
- Second Chapter is a background chapter that presents frequently used terms, a literature review on related researches, definitions, and technologies used in access control systems and a high-level compression between each of them.
- Third Chapter contains the functional specifications of the project, implementation details, and the evaluation.
- Fourth Chapter provides a critical discussion and suggestions for future works.

- Fifth Chapter provides a conclusion describing the work that has been done and contributions that have been made to the thesis project.

1.6 Project Source Codes

The links for developed tools can be found on following links.

The 'README.md' file in each project contains the instructions to build and deploy the tools.

Semantic Reasoner:

- https://github.com/ugurbayram/sabac

PDP Request Preprocessor:

- https://github.com/ugurbayram/pdputils-custom-preprocessor

CHAPTER 2

Background

2.1 Access Control

Access Control can be defined as a security technique that regulates who or what can access which resource under certain privileges, such as, full access, limited access, or no access. In other words, it is any technique used to control passage in to or out of an area, property, application, portal and approving the use of resources and data under certain circumstances. It is a fundamental security concept that reduces the risk of organizations and businesses.

There are various types of access control. The most abstract types are physical and logical. Physical access control would restrict the access of a building, room, computer equipment, or physical IT assets, while logical access control limits the network connection, application login, system files, and data. When the number of users/employees increases in a company, it is often the case that the company will limit certain users to access certain files and data. It makes sense to let the employees access only those information that is related to their work and duties and restrict those that are irrelevant to their daily tasks such as financial statistics and sensitive customer information.

The most common way to establish a proper security layer, a user is given an id card that provides access to the physical assets and buildings. Along with that, a user account is created that lets users to enter certain portals and internal or external websites that are used for the company's day-to-day tasks. The user account is a digital identity that is linked to an individual and it is secured with a verifier that is known by user only. The verifier can be a password, a PIN, a biometric scan, a security token, etc. When a user enters to a company portal to access certain data with an account id or username along with the verifier, the access control mechanism authenticates the user. Some of access control systems provide multi-factor authentication which requires multiple authentication types to verify the user's identity. Authentication can be described as a technique used to verify that someone is who they claim to be. For instance, when a bank customer places the bank card into ATM

machine, the machine asks user to enter the valid pin code which is expected to be known by the owner only. If the customer enters correct passcode, the bank confirms the identity of the customer and approves the rightful use of the card by the customer. This authentication merely identifies who the customer is, but nothing else. However, authentication by itself is not sufficient enough to protect the data. There is a need for an additional layer called authorization.

Authorization is a process to determine the authenticated user has access to particular resources or services. It usually takes place after user identity is verified and it controls either the user should access the data or apply the transaction that they are attempting to do. For instance, in an organization, verifying employees' ID and passwords are authentication. Determining which employee can access which floor is authorization. Access control policies ensure users are who they say they are and they have appropriate access to the requested data or file. In nutshell, an attempt to access a resource might allow authentication by entering valid credentials, but it must only be accepted if the authorization also completes successfully. If the attempt of authentication is succeeded but the authorization is failed, the access to the resource must be denied by the access control system.

2.1.1 Discretionary Access Control (DAC)

In DAC, each resource has a clearly identified owner. The owner of a resource can decide at his discretion to allow other users or subjects access to that resource. The resource owner is the main characteristic of DAC. The access control policies are defined as high-level directives (rules) that specify who (subject) has permission to practice what (action) on which (object) [8]. There are three fundamental concepts that can be extracted from this definition:

Subject: The active entity that accesses to data. This can be a user, an application or an IP address.

Object: The passive entity which a subject tries to access such as a file or a directory on a server.

Action: The action that a subject applies on an object. There are three types of actions: **R**ead, **W**rite, **O**wn.

DAC is implemented by creating an access control list which is associated with each object. The owner has full control of the object and can decide which other users or groups can access. Table 2.1 illustrates the DAC concept in brief. As shown in the first row, subject-1 is the owner of object-1. This means subject-1 can provide other subjects certain access on object-1. Whereas subject-1 can review and modify object-2 with given access type by subject-2 who is the owner of the object-2. However, subject-1 can only review the content of object-3 as the owner has provided only read access.

Table 2.1: The subject, object, and rule matrix

	object-1	**object-2**	**object-3**
subject-1	Read Write Own	Read Write	Read
subject-2	Read	Read Write Own	Read Write
subject-3	Read	Write	Read Write Own

As an access control policy DAC is determined by the object's owner or by someone other users who was granted access to that object. This allows owner to grant access based on the need-to-know bases to a user who can perform the following four tests: pass information, grant privilege to other users, refactor security settings for other users, and change rules for access. DAC is commonly used in well known operating systems and database systems. However, it is known as the weakest access control system due to several constraints it inherits from being strictly bound to one owner user and that user can transfer authenticated access to other users.

By being one of the least restrictive access control models, DAC is appropriate for small companies and organizations because it offers fast authorization, minimal administration obligations, convenience, and simplicity in terms of implementation and maintainability. However, for larger companies that have hundreds of employees, this access control model has certain drawbacks such as lack of complexity, termination control, and onboarding.

2.1.2 Mandatody Access Control (MAC)

Mandatory-access control (MAC) is a well-established model in the computer security. Despite the fact that it lends itself well to military environments, it represents clearly distinguishing aspects in controlling information flows. Such information flows are foremost characterized as being deterministic [9]. In MAC systems, user privileges are not resource owners or product owner oriented. MAC does not contain any ownership concepts. It is based on a hierarchical level. The resource owner does not have a right to decide who may access the resource. Instead, access is decided by a group of users who have the complete authority to set authorization rules.

As opposed to DAC, where the users could make changes to the permissions, in MAC once the administrator sets up the permissions nothing can be changed. It leads to much stricter control of the resources. It is based on the concept of

multi-level security which is widely used in military organizations. For instance, while defining a distinguishable level of security as below:

top-secret > secret > confidential > restricted > unclassified

Given that we define different levels of security, in which one is greater than the other in terms of security, we continue to have subject and object pairs that fulfill the workflow. Subjects are the users and objects are the resources that are aimed to be controlled. The subject is specified to have a security clearance at a particular level and the object is classified at a particular level. For instance, the user could be given a clearance at the level of secret where resource is classified as confidential or unclassified which is the lowest level in the above example. The security administrators would define security clearances and the classifications, which are two properties that are required to set up the confidentiality of the system upfront.

No read up: A subject can only read an object of a less or equal security level. **No write down**: A subject can only write into an object of a greater or equal security level.

A user on a secret level with 'no read up' and 'no write down' rules can read any documents that are lower and equal to 'secret', e.g., confidential, and can only write to higher or equal levels such as secret and top-secret. The main purpose of having these two rules together is to avoid releasing higher level confidentiality to lower levels.

2.1.3 RBAC

RBAC is an access control mechanism restricting system access relying on subjects roles. Users are authorized to access a resource or file by being assigned to specific roles. When an organization assigns a role-based access control to its employees, defined roles determine what kind of permission a user is granted. For instance, a given role may define the user type, such as administrator, specialist, or an end-user, and may limit access to certain resources or tasks. An organization may let certain individuals create and modify files while limiting others with view permission only.

RBAC is considered as a much more generalized model than MAC and DAC [10]. The researchers [10] state that the other solutions are policy-neutral frameworks while RBAC can be customized per-application basis. This provides flexibility to security admins to define variety of restrictions and access rules. Unlike DAC and MAC where subjects have some rights on objects, in RBAC users are assigned to roles and those roles have some rights on objects.

RBAC offers a great extend of abstraction at an enterprise business level rather than at a user identity level. The basic role concept is straightforward: establish access decisions and permission-based on functional roles in an enterprise

organization and then link the users to a role or a set of roles. Generally, roles stand for tasks, individual responsibilities, and qualifications in an enterprise. As shown in Figure 2.1, user groups are tied to certain roles, and the roles are associated with related permissions, i.e., related applications. User groups can be linked to one or more roles, just like the roles can be assigned to multiple permissions or tasks. The user's permissions list is equal to the combination of the permissions that user roles are linked to. Users can be reassigned to any other role without modifying the underlying access structure (i.e., policy). The structure is designed as a hierarchy between the user, role, and permissions. RBAC is, therefore, more flexible and scalable compared to other user-based mechanisms. Knowing that the turnover and task reassignments happen regularly in an enterprise, RBAC provides a stable and less complex mechanism to maintain user access permissions and avoid potential administration errors that might occur due to overlapping permissions. Although RBAC started to be

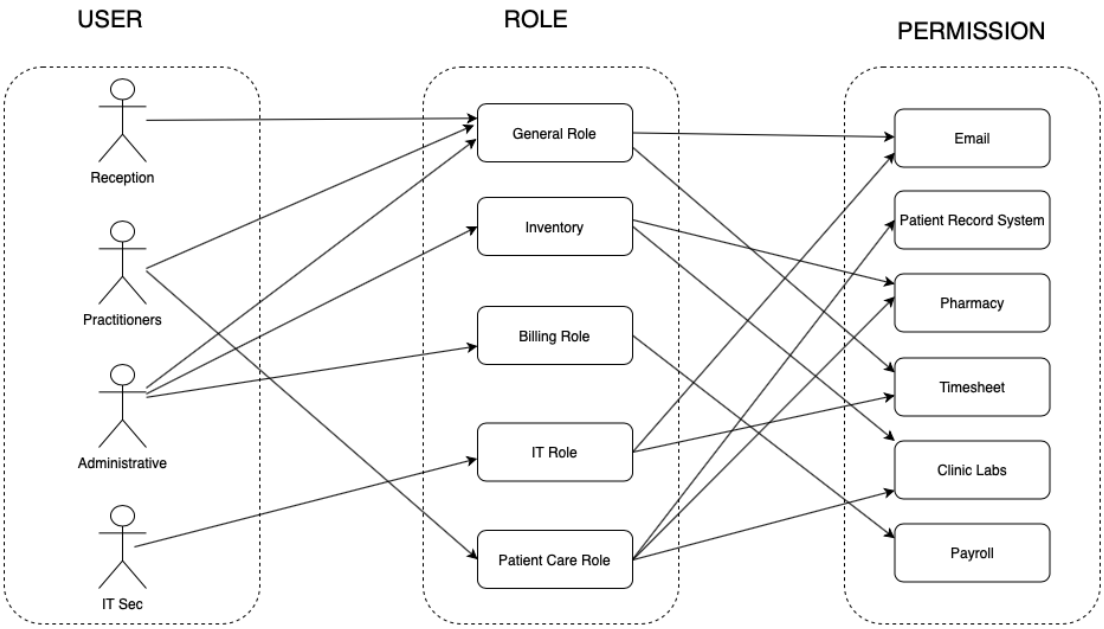

Figure 2.1: RBAC sample drawing

used in IT early 2000s, the same basic concepts have been used a few decades before in terms of managing the privileges. However, every organization implemented the roles in different ways which then led to lack of certain standards. NIST discovered this need and provided detailed analyses at [11].

According to NIST, there are four different models of RBAC that might help to increase functional capabilities of the organizations. They are Flat RBAC, Hierarchical RBAC, Constrained RBAC, and Symmetric RBAC. In general, these modes are suggested to have principle of least privileges, task, and duty separation. The least privilege principle is maintained by configuring RBAC assigning the privileges that are needed for a specific task. Task and duty separation is maintained by assigning two mutually exclusive roles to complete a task.

While RBAC has extended flexibility of configuring variety of users-roles-permissions, there are certain drawbacks of this access model. In initial state, where number of users, roles, and permissions are defined, it is trivial to set up the privileges. However, the IT infrastructure and number of employees in an organization keeps growing. Application permissions and user-role lists require regular revise causing cumbersome administrative tasks to maintain and manage IT Security which impacts scalability and dynamism of the IT infrastructure.

Despite the above-mentioned advantages, RBAC has the following drawbacks:

Role Explosion: RBAC does access permission through the subject roles. In many case, one user is assigned multiple roles. Due to that, when number of people increase in an organization, it causes an exponential increase on number of roles to due to the fact to accommodate different permission combinations. Each user often require unique access rights. On one hand, assigning generic roles to all group of user ruins the flexibility. On the other hand, assigning multiple roles to all users cause so many combinations to handle. [12] **Toxic Combinations:** Assigning multiple roles to users does not only increase the manageability but it also causes conflicting data. For instance, a role might allow user to create a purchase order, while another allowing the user to approve it. In this context, the role assignments posse a significant business risk if not managed properly. [12]

Lack of Context: As described in earlier sections, role definitions and assignments in RBAC is static. Therefore, it cannot offer policies that depend on contextual details such as time, location, subject relationship, etc. In other words, RBAC is not able to determine relationships between users and using that information to make policy decisions. RBAC was originally designed to answer just one question: "What access does a user have based on their assigned role(s)?" [12]

2.1.4 ABAC

ABAC is a further evolution of access control models to adapt to organizations that need a more advanced access control mechanism. ABAC takes into consideration other factors besides the identity and role. Examples of these

factors are the users' location, time, temporal constraints, the level of risk, etc. ABAC describes an access control model where the authorization decision is based on attributes that are designed and assigned to subjects and objects by combining environmental variables together with a set of policies linked to these attributes. Attributes are defined as characteristics that belong to both subject and object. Examples to subject attributes can be user ID, name, role, organization, nationality, where object attributes can be name, owner, data creation date. Environment conditions are contextual information associated with the access request.

With ABAC, access is assigned based on attributes or characteristics about the subject making the access request, about the file or object being requested, about actions, and about the environmental conditions. Granular policies can be established based on a combination of these attributes to grant or deny an access.

Researchers in NIST article [13] explain the evaluations from Access Control List (ACL) to Identity Based Access Control (IBAC) to RBAC and finally to ABAC and provides in-dept overview of the evaluation of Access Control Models. According to NIST, ACL and RBAC are special cases of ABAC in terms of the attributes used. They argue that the key difference with ABAC is the concept of policies that express a complex Boolean rule which can evaluate many different attributes. Although it is possible to achieve ABAC objectives with ACL and RBAC, it is difficult and costly to demonstrate access control requirements with these models due to the required level of abstraction. Furthermore, another challenge appears when access control requirements change. It is cumbersome to discover the places where ACL or RBAC implementation need to be updated.

ABAC avoids assigning object/operation pairs directly to subjects or to their roles before a request is initiated. Instead, when a subject sends a request to access to a resource (i.e., an object), ABAC engine replies with approve, deny, or indeterminate by processing the subject attributes, object attributes, environment conditions, and a set of policies that are specified with those attributes and conditions. Access control policies can be defined without a direct reference to potential subjects and objects. Similarly, users and objects can be provisioned without any reference to policies. Table 2.2 compares the ABAC with traditional access control models.

ABAC implementation and use-cases are detailed in the following Section.

2.2 Attribute Based Access Control

The main objective of access control systems is to serve a specific purpose by following a simple workflow. A subject tries to access objects within the

Table 2.2: ABAC model vs traditional Access Control (AC) models

ABAC model	Traditional AC model
Dynamic – access control permissions are evaluated at the time of actual request is made	**Static** – access control permissions are predetermined
Contextual – environmental conditions may be considered	**Limited context** – environmental conditions are not fully considered (time, location, environmental roles, etc.)
Fine grained – attribute based, so detailed rules can be formed	**Coarse** – classification is done at high abstraction level

specified scope of work and workflow. The access control system checks if the subject has the necessary privileges and allows or denies an access. Methods followed by the access controls to check the privilege, and differences from each other were described as high level in the previous sections. While non-ABAC systems give permissions based on a role, identity, and/or similar static control elements, ABAC systems give them based on several dynamic attributes and environmental conditions. In this section, we will elaborate on the intermediate elements, methods, and workflows that ABAC uses to make a decision.

We, as people, live in a multi-dimensional world. Our decisions, acts, and behaviors are based on several different dimensions. Idea of ABAC comes from this phenomenon saying "AC should take multi-dimensional characteristics to serve a purpose": Identity of the user, information, and characteristics of data that we are trying to access, and a lot of contextual information about when access is appropriate and when it should be denied. ABAC takes all these dimensions into consideration while making an access decision. Figure 2.2 indicates such an intersection of *Identity*, *Data* and *Context* dimensions. In fact, each of these entities can be related to different access control systems such as IBAC, RBAC and ABAC. Identity circle might stand for the IBAC. All what it cares is who the user is. Looking into the data circle, one might need to introduce roles and groups, which are the premises of the RBAC. In RBAC, we are not able to introduce the context. That is where ABAC comes into the picture and helps to bring those three dimensions together.

ABAC is a logical and distinguishable access control model as it controls the defined set of rules in order to provide access to an object. These rules combine entities (subjects and objects) attributes to environmental requests and operations. NIST defines ABAC as "Grant or deny user request based on arbitrary attributes of the subject and arbitrary attributes of the object and environment conditions that may be globally recognized and more relevant

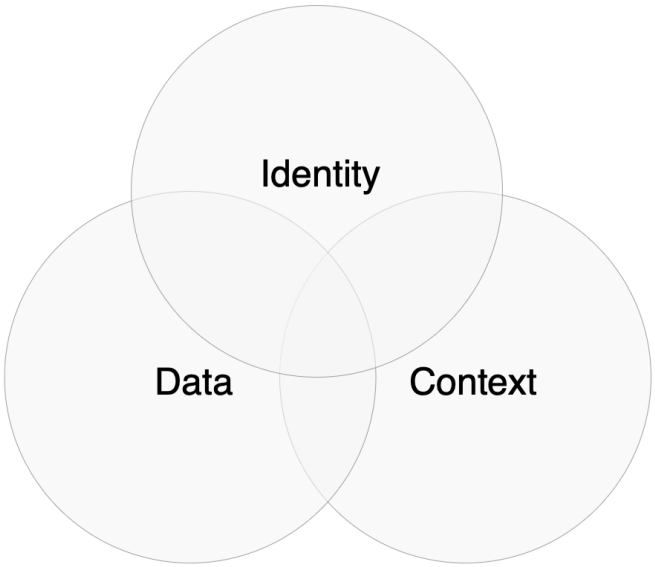

Figure 2.2: Multi dimensional AC

to the policies at hand". It is the combination that gives ABAC considerable power and flexibility. One important take away from the definition is "grant or deny". ABAC is not all about permitting access or giving privilege as IBAC and RBAC does, but also allowing user to define negative rule checks that make it possible to deny the privilege or in other words close the gates. Being able to deny rules can be very efficient in managing access control in certain environments than trying to permit rules based on user's rights.

Based on above definition, ABAC can grant or deny access based on combination of any attributes that allows system to set up a relationship. Below are some of those user and object attributes that are used.

- **Who** can access information
- **What** information can they access
- **When** can they access information
- **Where** can they access information from
- **How**, from which device or via which API can they access information
- **Why**, for what reason can they access information.

One of the key benefits and the strength of ABAC is the ability to naturally express relationships. It can be a direct relationship, for example, a doctor cares for a patient. Or it can set an indirect relationship, such as a user can view a document because the document and the user belong to same department. Being able to express a relationship does not only make ABAC powerful but also makes it easy to implement scenarios that otherwise would require dozens of roles.

Definition of subject, object, and resources was given in previous sections. ABAC system contains minor contextual differences while considering these entities. NIST [13] describes attributes as *"characteristics of the subject, object, or environment conditions"*. Based on this definition, we can describe ABAC entities as follow:

Subject is a user or Non-Person Entity (NPE) such as an application or a device demanding access to perform an operation on objects. Subject might be assigned several attributes.

Object is any kind of data that is stored in the network. It can also be a system resource for which access is managed by ABAC.

Operation is an execution of a function that subject applies in order to access an object. Operation attributes can be listed as read, write, edit, delete, copy, execute, and modify.

Policy is set of rules, relations, and combinations of state data that allows an authorization request.

Environmental Conditions is an operational or situational context in which access request should occur.

Figure 2.3 illustrates how an access request is processed in ABAC system:

1. Subject initiates request to access Object
2. ABAC takes policies, environmental conditions, subject and object attributes into consideration to evaluate an access request.
3. ABAC grants or denies the access based on the evaluation. Granted access might get revoked if any of entities attributes gets changed in time and evaluation returns false afterwards.

On one hand, ABAC is known successor of traditional AC models such as RBAC and IBAC where resources and subjects have attributes and a set of attributes can be understood as defining a role. On the other hand, ABAC enables a more precise AC model by allowing higher number of discrete inputs into AC decisions. Therefore, a larger set of possible combinations is brought

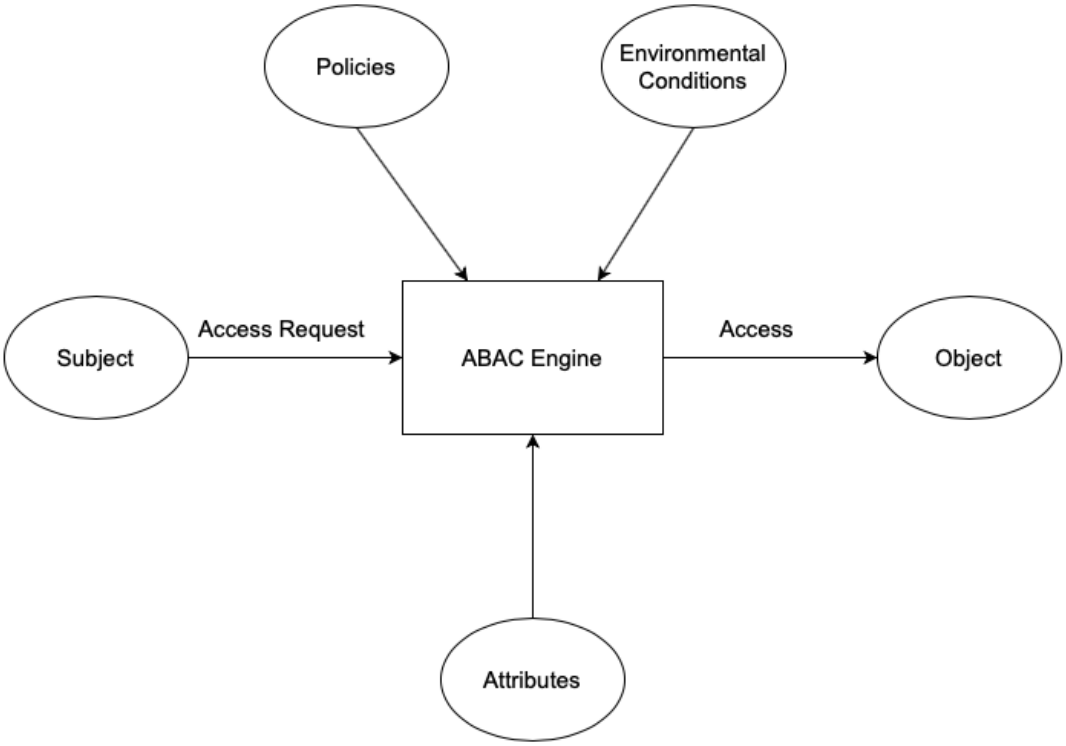

Figure 2.3: ABAC process

out to the system which makes it more flexible and comprehensive in terms of extensibility in different areas and domains. With this flexibility, AC system can create distinct and individual relationships and avoids binding the rules directly with objects and subjects.

In many organizations, big amount of information passes through different access control systems and is being exchanged between many technical and non-technical people. One good access control system should reduce security and privacy risks and share only the required information under the right circumstances with the right individuals. This is where ABAC steps in very elegantly. It allows the user to define policies that very carefully and very specifically define the right level of access control for the right messages, the right contracts, and the right circumstances. Policies are competitive advantages that come as part of ABAC system. It turns access control into understandable definitions and represents business logic rather than role-permission kinds of structures.

2.2.1 An Example of Fine-grained Access Control in e-Health

The Nurse example described in NIST article [13] is very much relevant to my master thesis. Hence, we find it necessary to describe it: A subject is assigned a set of subject attributes upon employment, such as Nancy Smith is a Nurse in the Cardiology Department. An object is assigned its object attributes upon creation, such as a folder with Medical Records of Heart Patients. Objects may receive their attributes either directly from the creator or as a result of automated scanning tools. The administrator or owner of an object creates an AC rule using attributes of subjects and objects to govern the set of allowable capabilities. For example **Nurses** in Cardiology department can **View** the **Records** of heart patients. Under ABAC, access decisions can change between requests simply by altering attribute values, without requiring changes to the subject/object relationships defining the underlying rule sets. This provides dynamic access control management capability to some extend and limits long-term maintenance requirements of object protections. Another policy can be designed as follow: **Nurses** can **View** the records of **Patients** in the same **Department** they are assigned to. This policy serves wider range of employees not only working in Cardiology Department but in all hospital, having similar privileges as in first example. Expanding such access control in ABAC requires small modification in policy while in other traditional AC models a harder work is required to not only modify object-subject roles and relations but also add new ones.

$$\underbrace{\textbf{Nurses}}_{\text{subject}} \text{ can } \underbrace{\textbf{View}}_{\text{action}} \underbrace{\textbf{Patients' records}}_{\text{object}} \text{ in the same } \underbrace{\textbf{Department}}_{\text{relationship}}$$

Given policy is shortened in order to simplify highlighting the attributes. The highlighted words in bold represent the attributes. The first one is *"Nurses"*. It is the subject who claims to have access to a resource. That would be user role in an IBAC system. Second one is *"View"*. It is a type of operation that subject wants to apply. Following attribute is *"Patients' records"*. It represents the object that the organization wants to protect or deals with business transactions. The fourth attribute is *"Department"*, or in other words, "same department" that stands for a relationship. It constructs a relationship between users department and records department, which can be seen as breaking down the authorization requirement into an authorization policy that uses the user's department as subject attribute on the one hand and the records as the object department (object attribute) on the other hand and compares the two together.

ABAC uses a specification language called eXtensible Access Control Markup Language (XACML), an XML-based language for access control that was standardized by Organization for the Advancement of Structured Information Standards (OASIS) consortium, to implement constructed policies and defined

attributes. This is an integration process that brings ABAC into use and enables communication between different applications to process authorization requests. XACML contains three parts. First one is the 'policy language' that defines how to express authorization constraints. Second one is the 'request/response scheme' that describes how to send authorization requests and get authorization decisions back. Third one is the 'architecture' which has a hierarchical structure to overview inner details.

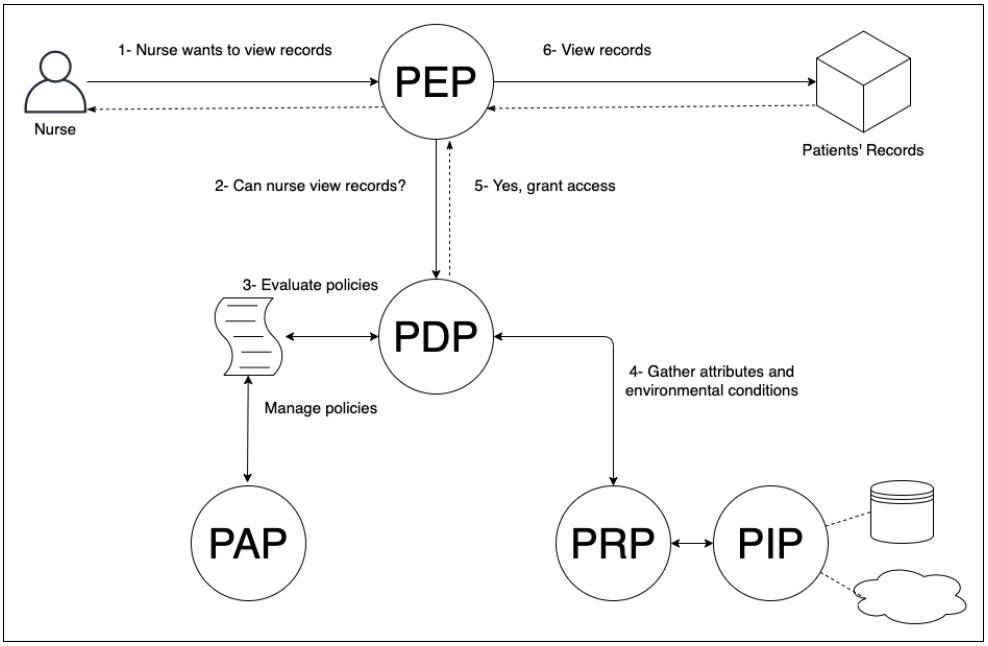

Figure 2.4: The XACML Architecture

Figure 2.4 illustrates the XACML architecture. As shown in Figure 2.4, there are five main components in the XACML architecture. From top to bottom, the first one is the Policy Enforcement Point (PEP). The enforcement point is responsible for protecting applications and resources. The application can be anything from a web service application to high-end application like SharePoint. PEP sends an authorization request to Policy Decision Point (PDP). The decision point is the core of the architecture, which loads policies and evaluates the policies against the incoming authorization requests. PDP eventually makes a decision and returns back to PEP. In order for the PDP to be able to make decisions, it needs to load XACML policies. These policies are stored in Policy Retrieval Point (PRP). In addition to policies, the decision point may need additional information about the authorization context. That information is retrieved from Policy Information Point (PIP). These information points can be databases that contain product or document information, LDAPs, active directories, where subject's attributes can be retrieved. Policy Administration Point (PAP) is the point through which administrators can define and write

authorization policies.

A basic ABAC flow for NIST nurses use-case is illustrated in Figure 2.4. On the top left corner, the nurse requests access to a document. PEP intercepts the requests and sends an authorization request to the PDP. PEP is responsible for converting a business request into an authorization request. The identity of nurses is determined by the enforcement point through the use of the authentication mechanism already in place. PDP loads the XACML policy from PRP and evaluates the received request against the policy. For the sake of simplicity, assume that only *"Nurses can view patients' records in the same department"* policy is retrieved. In order for evaluation to take place, PDP requires additional information regarding the nurse and patients' records. These subject and object-related attributes are retrieved from PIP and added into evaluation process. The simplest evaluation for our use-case is to compare if nurse's department is equal to record's department. PDP makes a decision as the result of these values read from different PIP. The decision is then forwarded back to the PEP. If the decision is 'Yes, grant access', then PEP enables the access of the records to the nurse within a given context.

The following provides advantages of ABAC:

Externalized authorization: In contrast to other access control systems, the business logic for authorization is decoupled from the application itself. Usually, access request is sent with a XACML request or a JSON request over Hypertext Transfer Protocol (HTTP)

Centralized authorization: Many organizations prefer installing ABAC in a centralized unit, usually in cloud. With the centralized approach, the authorization logic is moved into a single point of management and that brings certain benefits to the organization, such as: easy maintenance and policy deployments, large hardware capacity, and reducing the costs.

Policy-driven authorization: ABAC is known being a policy driven access control, in which the authorization logic is expressed as configurable policy definitions rather than writing the code.

Attribute-based access control: Policies use attributes are the building blocks for ABAC system. Attributes can be anything related to subject, object, access type, even the environmental conditions. Policies are created by setting certain relationships between attributes. Compared to other access control systems, ABAC is extremely flexible in constructing dynamic policies.

Along with listed advantages of ABAC system, there are certain drawbacks due to system complexity and high-level granularity. Table 2.3 provides a comparative research made by [6] listing a feature comparison between the two popular access control models.

Table 2.3: Feature comparison between RBAC and ABAC

Issues	RBAC	ABAC
Trend in 2018	Medium	High
Global Agreement	No	Yes
Flexibility	No	Yes
Easiness	Yes	No
Dynamicity	No	Yes
Authorization Decision	Locally	Globally
Granularity	Low	High
Manageability	Simple	Complex
Conviction	Locally	Globally
Confusing deputy	No	Yes
Changing privileges	Complex	Simple
Role explosion problem	Yes	No

Despite several advantages of ABAC, it is not the best solution in distributed environments in which several entities from different domains collaborate with each other. In such environments a mismatch between the name of attributes is inevitable. Hence, there is a need to extend the ABAC with semantic technologies.

2.3 Semantics Technologies

In IoT world, there exists a huge diversity of system, device, platform, and environment which makes interoperability of these components more and more complex. The handbook [14] defines semantic technology as: "Semantic technology provides machine-understandable or better machine-processable descriptions of data, programs, and infrastructure, enabling computers to reflect on these artifacts. In other words, it is to create data structures which machines can easily read, understands, and integrate them into existing systems." For instance, when a IoT smart sensor is added to the system, it should send the information using a common language which should not matter which platform or system it is connecting with. The platform should be able to take that information and process it in a way that fits the purpose. However, creating a common language is not easy to achieve with current real-world scenarios where there exist thousands of IoT service providers globally with different systems, platforms, device portfolios, with different business focus. Therefore, it is required to do certain information engineering, and data engineering to make it effectively be able to interoperate between different platforms and systems. Semantic technologies try to address this by providing data modeling and offering certain representation frameworks in order to be able to address

to create more structured data and to be able to represent them in a way that machines can read and understand. Although XML seems to offer such data modeling that helps to solve interoperability problems to some extend, one should know that it is limited to offer hierarchical information and it does not have ontological commitment. Hence, it can not serve more complex structures such as time-series data or where we require to know relationship between different pieces of data.

One of the key frameworks for representing semantic data is Resource Description Framework (RDF). The main difference between RDF and XML is that RDF represents data in graph format while XML represents in a hierarchical format. The notion of the graph in RDF is called 'triple'. The 'triple' stands for a subject, an object, and predicate. The predicate is nothing but a relationship between subject and object. A very simple example can be "Person has name". Where "Person" is subject, "name" is object and "has" is a relationship between subject and object. With that, the relationships have a label and this structure allows us to write a program to do a search over a big data set to extract information. In other words, the labels and relations set a path or structured link data that software programs can follow and extract different pieces of information from large graphs of data. Such structured data link can be effectively used in IoT world. For instance, for temperature sensors sending data in different units like Fahrenheit and Celsius to a portal can have a relationship as "hasUnitOfMeasurement". And then, a software program can run a search to find the sensor data by this relationship, extract all sensor data in Fahrenheit and finally run a conversion function changing them to Celsius. By that, all sensor data can be standardized as Celsius.

2.4 Ontology

The term ontology is originated in Computer Science from philosophy and in that context, it is used as the name of sub-field of philosophy, namely, the study of the nature of existence. In semantic technology, an ontology is a formal specification of a domain; concepts in a domain and relationships between the concepts and some logical restrictions. Despite many complex definitions of ontology, the main idea of ontology in semantic technology is to create a common vocabulary and common concept for a domain. Common concept stands for common labels to present these concepts and their relationships. There are two main parts in this process involving different groups of people. One part is knowledge engineering, and the other part is working with domain experts who are familiar with the domain and the concepts int the domain. For instance, a simple university domain consists of different concepts such as students, courses, staff members, modules, lecture theatres, and schools. In terms of IoT, using sensors in health monitoring domain, the concepts would consist of person, person name, sensor accuracy, sensor manufacturer, sensor

location, sensor measures, etc. These concepts and relationships generate a model.

Like semantic technology, ontology has different definition languages to represent the concept in a machine-readable format. The most common one is Web Ontology Language (OWL). It is a logical definition language which is based on formal logic and description logic that allows us to represent the concept and the relationships and add some constraints if needed. OWL provides more concepts to express meaning and semantics than eXtensible Markup Language (XML) and RDF and it provides more constructs for stating logical expressions such as Equality, Property Characteristics, Property Restrictions, Restricted Cardinality, Class Intersection, Annotation Properties, Versioning, etc...

2.5 IoT Gateways

IoT is surrounded by a set of advanced equipment such as sensors and meters, smart devices and software, that helps to exchange the information between devices and results the interoperability among them. IoT technology holds significant potential in the overall IT and communication industry not only in developing countries but globally. Therefore IoT technology market is gaining rapid growth due to the rising adoption of new technologies and trends.

By providing digital transformations and empowering the upgrade of an existing process and business model, IoT technology has been the keystone for several organizations. The key players in the IoT market are developing strategies to invent new advanced IoT products and solutions. Many enterprises in different sizes are immensely adopting IoT solutions in order to increase the cost efficiency, productivity, and operation enhancements in day-to-day business. Furthermore, the rapid adoption of cloud-based solutions in the IT industry is acting as the key driver for the growth of the IoT market during the forecast period.

2.5.1 Raspberry PI

The Raspberry PI (RPi) is one of successful open source projects introduced by Raspberry Pi Foundation, a registered educational charity based in the UK. Their ultimate goal is to promote teaching computer science essentials and advance education of adults and children, mainly in the field of computer and related subjects. They have released RPi as a commercial product in 2012. RPi is a low-cost, powerful single board computer with approximately size of a credit card. The device has gone through several upgrades that improved the capability and reduced the production cost. Different versions of the device have been sold from 10 USD to 40 USD. As being a low-powered portable device, the fame of RPi went far beyond than anticipated. RPi has been used

in a wide variety of helpful projects, from powering home-made robots to serving as home theater PCs.

Since the day it was introduced, RPi has been continuously improved the computation capability by upgrading processor power, storage, and other hardware features. Newer models of the device have four USB ports, a full-sized ethernet jack, and up to 4GB of memory, quad-core processor. The real reason that the RPi is popular is the ability to configure input and output pins that come with the device. General Purpose Input Ouput (GPIO) pins can receive electrical signals and transform them into digital signals that the operating system can understand. The ability to control electrical signals enables users to control a lot of small hardware equipment that uses electricity, such as LED lights, buttons, low powered motors, switches, radio signals, audio signals, and even small LCD display panels. By using GPIO pins one can take RPi from the computing world to the physical world. The following provides the full specification of RPi model 4B extracted from [15]:

- SoC: Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- GPU: Broadcom Videocore-IV
- RAM: 2GB, 4GB or 8GB LPDDR4-3200 SDRAM
- Networking: 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
- Bluetooth: Bluetooth 5.0, Bluetooth Low Energy (BLE)
- Storage: Micro-SD
- GPIO: 40-pin GPIO header, populated
- Ports: 2 × micro-HDMI 2.0, 3.5mm analogue audio-video jack, 2 × USB 2.0, 2 × USB 3.0, Gigabit Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensions: 88mm × 58mm × 19.5mm, 46 g

RPi shares much in common with a desktop PC. It has a Broadcom CPU which is connected to the RAM, external storage, and several other ports on the board. It runs a free Operating System (OS) called Raspbian that handles the computer's basic functions, runs programs, controls the HDMI, USB and RJ45 ports, and renders a graphical interface. Rather than a brand new OS, Raspbian is a modified version of the popular Debian. It is a patched version of the Linux Kernel, which can be found on the Raspberry Pi GitHub repository. Raspbian adds several RPi optimizations to the kernel sources. Although the

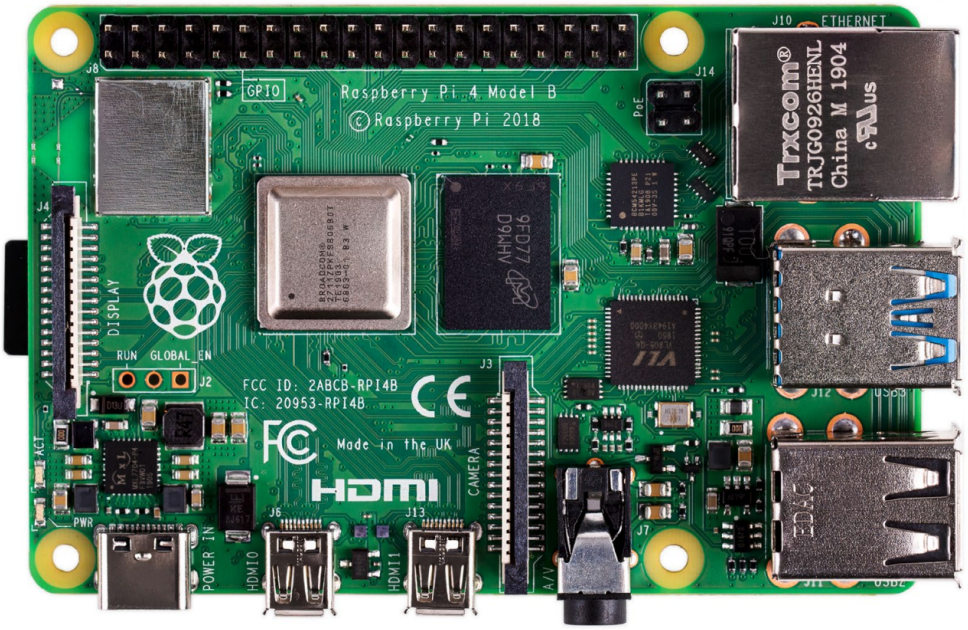

Figure 2.5: Raspberry Pi Model 4B

OS is designed and optimized for RPi devices, it is not limited to installs on other computing devices.

As a low-cost, small sized, portable yet powerful device, RPi has an increasing popularity in many IT industries. The Raspberry Pi Foundation, related community forums, and the internet provide a wealth of information on RPis different uses, alongside tips on implementation of different applications and projects. The followings are examples highlighting the range of possibilities where this micro-computer can be used: Home automation system, simple robotics, machine learning, gaming, surveillance camera system, and network tools such as VPN server, gateway, router, etc. However, there is no doubt that IoT is one of those that gain the favor the most. According to the book [16] as a single chip computer like RPi can be used as a strong device in controlling applications and appliances with the internet and therefore it forms an ideal platform for low-cost IoT based applications. They also stated that the RPi is still in its growing stage, although it is evolving very fast every day, and comes at a very low cost, the trade-off is pretty much reasonable.

2.5.2 Arduino

Arduino is another interesting innovation that was originally started as a research project at Interaction Design Institute of Ivrea in Italy. The very first board was introduced in 2005. The goal was to help design students, who did not have previous hardware and micro-controller programming experience to

build certain working prototypes interconnecting physical world to digital world. That makes Arduino one of first widespread open-source hardware project aiming of building a self-reinforced community. They pledged themselves to spread the use of such hardware tools with contribution of group of people all around the globe. Some of their duties are to help to debug existing code, writing new code examples, creating tutorials and attracting other people to join the community. Since the first board was introduced, there has been a massive evolvement with many new development boards and software libraries being introduced and expanding the range of possibilities available to the community.

Arduino enters to the market as an open-source prototyping board with Atmel microcontrollers. A microcontroller is a circuit board with one or more processing units with internal memory and programmable input/output peripherals. As indicated in technical reference book [17], it is essential to notice that a microcontroller is not a general-purpose CPU. Microcontrollers are typically used for a limited set of specific tasks, such as sensing keypress actions in a keyboard, controlling some motors, monitoring temperatures, or providing the "smarts" for a dynamic art project or all of that at the same time. Microcontrollers are used in wide range of consumer products including electronic toys, automobiles, office machines, households, etc. Some of those may not require sophisticated hardware specifications and may have minimal requirements for memory and programming length and low software complexity. On one hand, a typical microcontroller is a self-contained device with only a limited amount of program memory and very little RAM, but it has many built-in I/O functions. On the other hand, embedded controllers come with very specific software programs for the need and can be maintained by vendor only. Arduino provides an easily programmable microcontroller that can be used with different purposes and it can easily be connected to breadboards and augmented with specialized expansion boards called shields. Since Arduino is open-source, there are numerous variants of boards with different designs and capabilities that serve different needs. The most popular one is Arduino UNO Rev 3, which is based on Atmel's ATmega328 microchip with the following specifications:

- Operating Voltage: 5V
- Recommended Input Voltage: 7-12V
- Input Voltage Limit: 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA

- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB of which 0.5 KB used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz
- Dimensions: 68.6 x 53.4 mm, 25 g

In contrast to RPi, Arduino has limited onboard memory of 2KB devoted to a special piece of firmware named bootloader. Therefore, there is a major difference in how Arduino handles the execution of requested tasks. As it is not feasible to develop, test, and run a code directly on the device with such internal memory, Arduino requires an external IDE to let developers write the code, compile, and uploaded onto the Atmel chip before it runs. Developers may install Arduino IDE on popular operating systems like Windows, Mac OS, Linux, or prefer using an online IDE platform. The IDE lets developers write the code in C and C++ programming language, compile the software package, and turn into machine-readable instructions. The compiled code is then deployed on the Arduino board to execute the commands. Arduino board works with low-level programs while Arduino IDE offers the flexibility of developing with high-level programming languages.

Arduino has been a popular product for IoT as well as a successful tool for education in field of science, technology, engineering, and mathematics. As stated in [18], hundreds of thousands of designers, engineers, students, and developers have been using Arduino to innovate in music, games, toys, smart homes, farming, autonomous vehicles, and more.

CHAPTER 3

Use Case Implementation

3.1 Project Baseline and Resources

The thesis purpose is set to provide a solution that integrates ABAC engine with a third party semantic reasoner that runs on a residential gateway. During the review and analysis phase of the thesis, we evaluated FIWARE Authzforce solutions to see if they would address our needs. FIWARE is an Open Community whose members are committed to materialize the FIWARE mission, that is *"to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors"* [19]. The community is funded by European Union with the vision of shaping Europe's digital future [20]. They are dedicated to implementing opensource IoT solutions mainly for Europe but accessible for global use. Authzforce is a group of talented software engineers working on open source projects and IoT solutions inside FIWARE community [19].

One of important projects that FIWARE provides is the ABAC framework compliant with the OASIS XACML standard v3.0. It consists of an authorization policy engine and a RESTful (Representational state transfer based web service) authorization server. It was primarily developed to provide advanced access control for Web Services or APIs. However, it is generic enough to address all kinds of access control use cases. As part of the thesis project, we have used the Java API project named Authzforce Core. It provides an XACML PDP engine as a Java library so that applications can instantiate and use an embedded XACML PDP easily with Java. Figure 2.4 in Section 2.2 illustrates the XACML architecture and describes how a policy request is evaluated in an ABAC engine. Following are the recommended [21] minimum hardware specifications that this tool would run without any issue.

- CPU frequency: 2.6 GHz min

- CPU architecture: i686/x86_64
- RAM: 4GB min
- Disk space: 10 GB min
- File system: ext4
- Operating System: Ubuntu 16.04 LTS

Java environment:

- Java Runtime Environment (JRE) 8 either from OpenJDK or Oracle;
- Tomcat 8.x.

With these technical specifications in mind, the RPi 4B with 4GB RAM model device seems to the best match for our needs. In addition to being a microcomputer, RPi runs on Raspbian operating system, which is derived from Debian Linux. Raspbian has taken more attention from the raspberry foundation given it is the official Raspberry OS. This results in the development of more features and utility software. With support from the raspberry community, it becomes easy to set up this distribution and get going. Raspbian comes pre-installed with useful tools such as office programs, a web browser, and some programming languages such as scratch, python, and C/C++. The Unix tools and ecosystem makes it quite convenient to install any other required software packages and programming environments, which in our case are JRE and Tomcat.

Figure 3.1 represents a high-level system architecture of the integration between ABAC engine and the Semantic Reasoner (SR). The diagram contains modification from Figure 2.3 on scenario text and integration points. During the thesis project implementation phase, the following simple scenario was focused: "Doctor wants to open patient's smart door lock if an emergency call is received and accepted within working hours". Converting that to more a policy kind of definition, it would look like as follows:

$$\underbrace{\textbf{Doctor}}_{\text{subject}} \text{ can } \underbrace{\textbf{access}}_{\text{action}} \underbrace{\textbf{patients' smart door lock}}_{\text{object}} \text{ in } \underbrace{\textbf{emergency state}}_{\text{environmental condition}}$$

$$\text{during } \underbrace{\textbf{working hours}}_{\text{environmental condition}}$$

In order for ABAC to understand and evaluate requests for this policy, it needs to be converted to XACML syntax as follows:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <ns5:PolicySet xmlns="http://authzforce.github.io/core/xmlns/pdp/6.0" xmlns:
       ns2="http://www.w3.org/2005/Atom" xmlns:ns3="http://authzforce.github.
       io/rest-api-model/xmlns/authz/5" xmlns:ns4="http://authzforce.github.io
       /pap-dao-flat-file/xmlns/properties/3.6" xmlns:ns5="urn:oasis:names:tc:
       xacml:3.0:core:schema:wd-17" PolicySetId="root" Version="1.0"
       PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
       algorithm:permit-overrides">
3     <ns5:Target/>
4     <ns5:Policy PolicyId="urn:oasis:names:tc:xacml:3.0:example:MyPolicyNew"
          Version="1.0" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-
          combining-algorithm:permit-overrides">
5        <ns5:Target/>
6        <ns5:Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:MyRule" Effect
             ="Permit">
7           <ns5:Target>
8              <ns5:AnyOf>
9                 <ns5:AllOf>
10                   <ns5:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                         string-equal">
11                      <ns5:AttributeValue DataType="http://www.w3.org/2001/
                            XMLSchema#string">doorLock</ns5:AttributeValue>
12                      <ns5:AttributeDesignator Category="urn:oasis:names:tc:
                            xacml:3.0:attribute-category:resource" AttributeId
                            ="urn:oasis:names:tc:xacml:1.0:resource:resource-
                            id" DataType="http://www.w3.org/2001/XMLSchema#
                            string" MustBePresent="false"/>
13                   </ns5:Match>
14                </ns5:AllOf>
15             </ns5:AnyOf>
16          </ns5:Target>
17          <ns5:Condition>
18             <ns5:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                   and">
19                <ns5:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                      string-at-least-one-member-of">
20                   <ns5:AttributeDesignator Category="urn:oasis:names:tc:
                         xacml:1.0:subject-category:access-subject"
                         AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
                         subject-id" DataType="http://www.w3.org/2001/
                         XMLSchema#string" MustBePresent="false"/>
21                   <ns5:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                         function:string-bag">
22                      <ns5:AttributeValue DataType="http://www.w3.org/2001/
                            XMLSchema#string">doctor</ns5:AttributeValue>
23                   </ns5:Apply>
24                </ns5:Apply>
25                <ns5:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                      string-at-least-one-member-of">
26                   <ns5:AttributeDesignator Category="urn:oasis:names:tc:
                         xacml:3.0:attribute-category:action" AttributeId="urn:
                         oasis:names:tc:xacml:1.0:action:action-id" DataType="
                         http://www.w3.org/2001/XMLSchema#string"
                         MustBePresent="false"/>
```

```
27                  <ns5:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                        function:string-bag">
28                     <ns5:AttributeValue DataType="http://www.w3.org/2001/
                           XMLSchema#string">access</ns5:AttributeValue>
29                  </ns5:Apply>
30               </ns5:Apply>
31            </ns5:Apply>
32         </ns5:Condition>
33      </ns5:Rule>
34      <ns5:Rule RuleId="Deny-Rule" Effect="Deny"/>
35   </ns5:Policy>
36 </ns5:PolicySet>
```

Listing 3.1: An XML Policy

NIST researchers [13] argues that *"**ABAC is a logical access control model** that is distinguishable because it controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment relevant to a request."*. However, with an out-of-the-box ABAC engine, which does not have integration with PRP or PIP, the policy evaluation is made based on given attributes inside the policy and request. In other words, if the requester for the above policy is a *"Physician"*, which is a similar title to *"Doctor"* in hospital management, or *"Fastlege"*, a family doctor in Norway, policy will deny the access on the object (i.e., patients' smart door lock). Another scenario can be a requester sending *"Open"* instead of *"Access"* as the action attribute. In this case, the request will be denied due to the mismatch between attribute values. Although the ABAC is surrounded with effective features that the industry needs, it certainly has a lack of context analyses over policy attributes. XACML architecture supports numerous functions to perform policy evaluation such as arithmetic functions, logical functions, string/numeric/non-numeric comparison functions. However, these functions are limited to perform logical operations and therefore have limitations on extracting the context from the attribute values.

3.2 System Architecture and High Level Design

The overall XACML architecture alongside ABAC flow is described in Section 2.2.1. Figure 3.1, indicates my contribution on the ABAC core with indicating the integration points. My contribution is highlighted with a red square.

Request Preprocessor Extension (RPE) is a simple plugin deployed in ABAC core library that enables intercepting incoming requests targeting the policy on PDP. As it intercepts the request before the evaluation process, it gives us a chance to review the request content and refactor attributes if necessary. As soon as the refactoring process is done, the flow resumes and the request

arrives at PDP for the final evaluation. The response is then forwarded to the requester.

SR tool is designed and implemented as a web application by using Java Spring boot, OWLApi, and Hermit. Although the semantic reasoner part of the application, which is to extract knowledge from an ontology, is an standalone offline process, receiving queries and providing responses, requires endpoints for web-based communications over HTTP Protocol. Restful web service is one of the most commonly used and well-suited way of creating Application Programming Interface (API)s. SR tool provides the API to let RPE initiates communication.

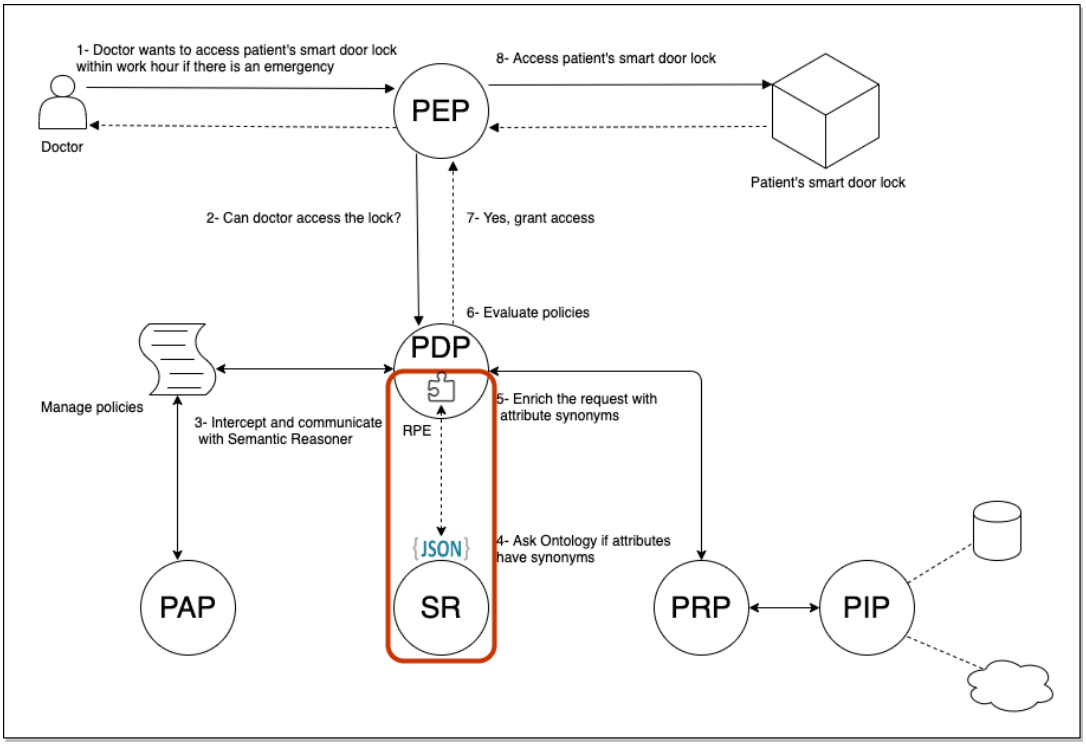

Figure 3.1: Semantic Attribute Based Access Control (S-ABAC) architecture drawing

Figure 3.2 is a basic process flow of the upgraded ABAC engine evaluating a policy request. A user targets a policy by requesting access on a certain resource. RPE intercepts the request and checks if it contains concerned attributes. The flow routes to PDP as a routine call before the upgrade if no attributes are identified. However, upon identifying relevant attributes in a request, the RPE sends a web service call to the SR tool by providing attribute values. Once the SR receives a request, it parses ontology and retrieves attribute synonyms and forwards the result back to the RPE. If the synonym list is not empty,

the RPE refactors the request by appending new attribute synonyms. The enriched request is then forwarded to the PDP for a final evaluation and user gets either 'Permit' or 'Deny' result after the evaluation.

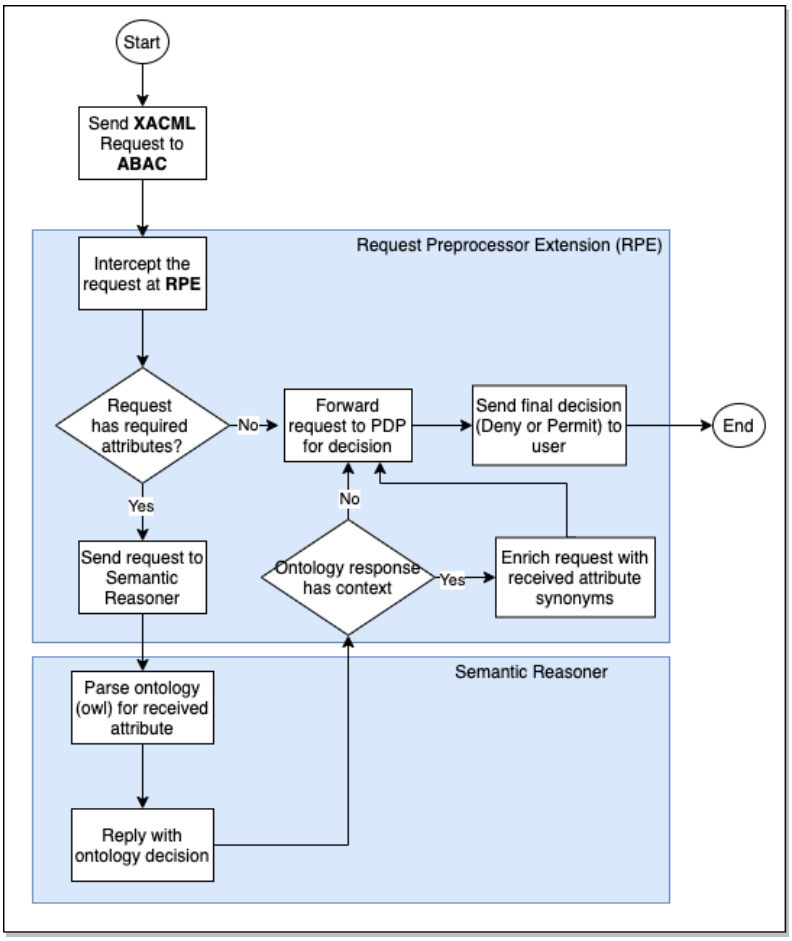

Figure 3.2: The proposed S-ABAC flowchart

3.3 Implementation

3.3.1 ABAC Engine Setup on Raspberry Pi 4

ABAC engine, or in other words Authzforce server is a heavy application developed in Java and has APIs that follow the Representational State Transfer (REST) architecture style and complies with XACML. The installation of ABAC engine requires Tomcat8 and JRE8. Following commands were run on the Raspberry Pi 4 Model B to install the Authzforce server.

To install OpenJDK8, run :

```
$ sudo apt install openjdk-8-jre
```

And to install Tomcat8 :

```
$ sudo apt install tomcat8
```

Authzforce team maintains ABAC engine up-to-date and provides the latest version in a global maven repository.

To download the latest Authzforce server:

```
$ wget https://repo1.maven.org/maven2/org/ow2/authzforce/authzforce
-ce-server-dist/8.1.0/authzforce-ce-server-dist-8.1.0.deb
```

The distribution package can be placed in desired directory to do the installation.

To install the Authzforce server:

```
$ sudo aptitude install gdebi curl
$ sudo gdebi authzforce-ce-server-dist-M.m.P.deb
```

Once installation is completed, tomcat should be restarted. To test the AutzhForce server exposed REST APIs, the following URL can be tested on a browser or 'curl'ed from a terminal.

```
$ http://localhost:8080/authzforce-ce/?_wadl
```

The installation of the ABAC engine initiates a default domain and sets a root policy that permits any XACML request. A custom policy can be added to the default domain or to a new domain. Once it is added and enabled, ABAC engine will evaluate requests that are targeted to the new policy set.

To create a custom policy set:

```
curl -X POST \
http://localhost:8080/authzforce-ce/domains/{domain-id}/pap/policies \
-H 'Content-Type: application/xml' \
-d '<PolicySet>...etc</PolicySet>'
```

As part of thesis project, we used the policy provided in Section 3.1. A tutorial [22] prepared by Authzforce team guides how to configure ABAC engine for basic needs.

3.3.2 Ontology and Semantic Reasoner Development and Deployment

Creating an ontology and parsing it on a semantic reasoner requires certain tools and software implementations. The thesis contains an ontological knowledge framework on a hospital scenario that has been created with Protégé editor. The output of the ontology is an OWL file. Furthermore, a semantic reasoner is developed to parse this ontology and extract knowledge out. The semantic reasoner tool is developed with Spring boot, Java, and Hermit Reasoner. The tool exposes endpoints to open a communication channel to outside world.

3.3.2.1 Protégé

Protégé is a free, open-source ontology editor and framework for building intelligent systems. It is supported by a strong community of academic, government, and corporate users, who use Protégé to build knowledge-based solutions in different areas such as biomedicine, e-commerce, and organizational modeling. Ontologies became central to many applications namely scientific knowledge portals, electronic commerce, and web services.

Ontology represented in Figure 3.3 is a minimal hospital scenario created for the semantic reasoner. 'Arc Types' panel indicates the colors and categories of arrows that stands for a relation between two entities. In our simple scenario, a hospital has staff who are dedicated to certain 'Departments' and have some 'Access' rights. There is a parent 'Role' class that has three dependent subclasses namely 'Role-Nurse', 'Role-Doctor', 'Role-Patient'. Similarly, 'Access' class has three access types such as 'Access-Delete', 'Access-Read' and finally 'Access-Write'. The prefixes are added in order to distinguish subclasses from individuals. In last lane of the hierarchy, there exist the individuals that are kind of synonyms for assigned subclasses, or in other words, for the roles. For instance, 'Lege', 'Doctor', 'Physician' stands for the same role doctor; where 'Read', 'Review', and 'Access' are individual synonyms for read access type.

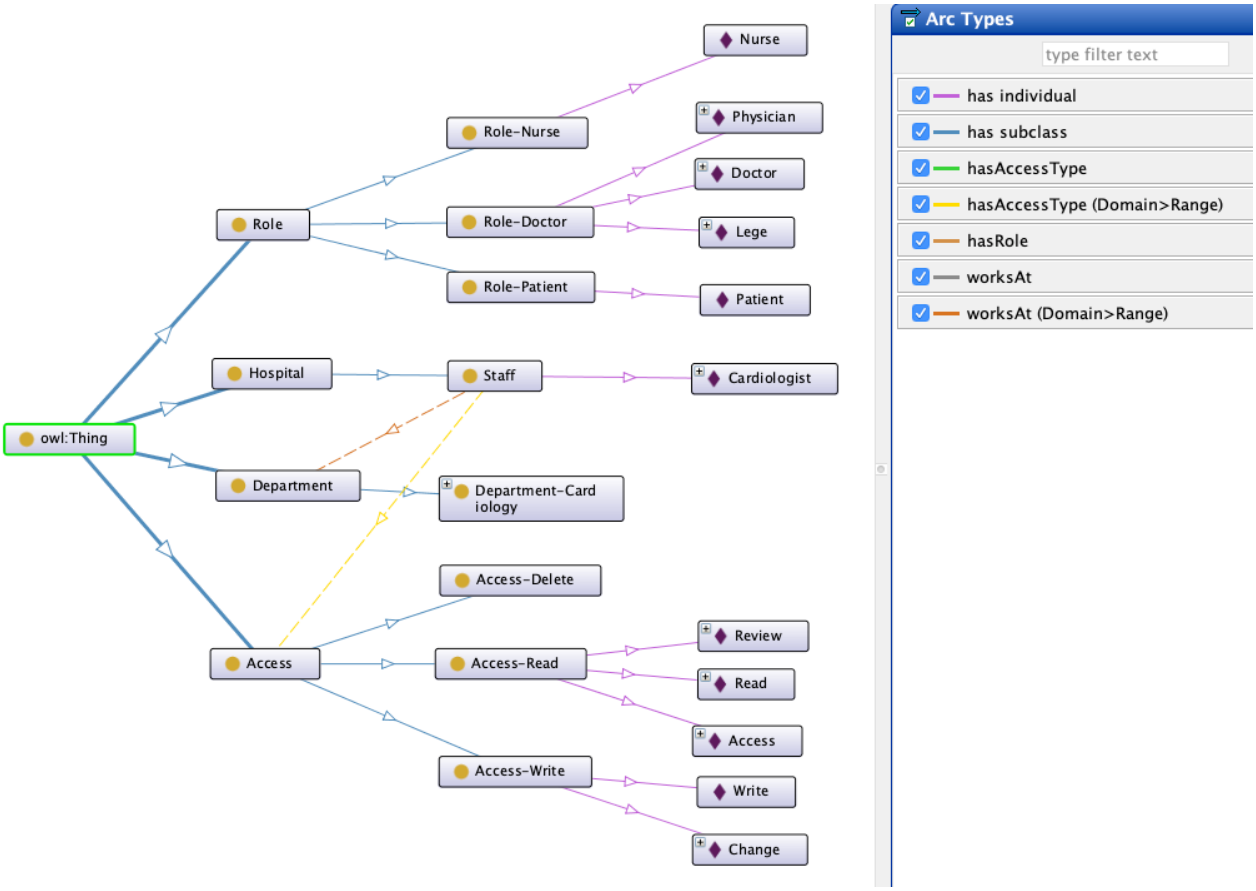

Figure 3.3: Hospital staff ontology

When exporting the designed ontology from Protégé editor, we get the OWL file as shown in Listing 3.2. This file contains formally specified components such as individuals (instances of objects), classes, attributes, and relations as well as restrictions, rules, and axioms. Class hierarchies, object properties, and individual relations compose an overall ontology. Semantic reasoner parses the framework and iterates through those relations in order to retrieve certain knowledge.

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns="http://ifi.uio.no/ontologies/health-role#"
3      xml:base="http://ifi.uio.no/ontologies/health-role"
4      xmlns:owl="http://www.w3.org/2002/07/owl#"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:xml="http://www.w3.org/XML/1998/namespace"
7      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
8      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
9      xmlns:health-role="http://ifi.uio.no/ontologies/health-role#">
10    <owl:Ontology rdf:about="http://ifi.uio.no/ontologies/health-role#"/>
11
```

```
12  <!--
13  ///////////////////////////////////////////////////////////////////////////////////
14  //
15  // Object Properties
16  //
17  ///////////////////////////////////////////////////////////////////////////////////
18   -->
19
20  <!-- http://ifi.uio.no/ontologies/health-role#hasAccessType -->
21
22  <owl:ObjectProperty rdf:about="http://ifi.uio.no/ontologies/health-role#
        hasAccessType">
23     <rdfs:domain rdf:resource="http://ifi.uio.no/ontologies/health-role#
           Staff"/>
24     <rdfs:range rdf:resource="http://ifi.uio.no/ontologies/health-role#
           Access"/>
25  </owl:ObjectProperty>
26
27  <!-- http://ifi.uio.no/ontologies/health-role#hasRole -->
28
29  <owl:ObjectProperty rdf:about="http://ifi.uio.no/ontologies/health-role#
        hasRole"/>
30
31  <!-- http://ifi.uio.no/ontologies/health-role#worksAt -->
32
33  <owl:ObjectProperty rdf:about="http://ifi.uio.no/ontologies/health-role#
        worksAt">
34     <rdfs:domain rdf:resource="http://ifi.uio.no/ontologies/health-role#
           Staff"/>
35     <rdfs:range rdf:resource="http://ifi.uio.no/ontologies/health-role#
           Department"/>
36  </owl:ObjectProperty>
37
38  <!--
39  ///////////////////////////////////////////////////////////////////////////////////
40  //
41  // Data properties
42  //
43  ///////////////////////////////////////////////////////////////////////////////////
44   -->
45
46  <!-- http://ifi.uio.no/ontologies/health-role#accessTypeName -->
47
48  <owl:DatatypeProperty rdf:about="http://ifi.uio.no/ontologies/health-
        role#accessTypeName">
49     <rdfs:domain rdf:resource="http://ifi.uio.no/ontologies/health-role#
           Access"/>
50  </owl:DatatypeProperty>
51
52  <!-- http://ifi.uio.no/ontologies/health-role#roleName -->
```

```
53
54    <owl:DatatypeProperty rdf:about="http://ifi.uio.no/ontologies/health-
          role#roleName">
55       <rdfs:domain rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Role"/>
56    </owl:DatatypeProperty>
57
58    <!--
59    ///////////////////////////////////////////////////////////////////////////////////////

60    //
61    // Classes
62    //
63    ///////////////////////////////////////////////////////////////////////////////////////

64     -->
65
66    <!-- http://ifi.uio.no/ontologies/health-role#Access -->
67
68    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Access"/>
69
70    <!-- http://ifi.uio.no/ontologies/health-role#Access-Delete -->
71
72    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Access-
          Delete">
73       <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
             role#Access"/>
74    </owl:Class>
75
76    <!-- http://ifi.uio.no/ontologies/health-role#Access-Read -->
77
78    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Access-
          Read">
79       <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
             role#Access"/>
80    </owl:Class>
81
82    <!-- http://ifi.uio.no/ontologies/health-role#Access-Write -->
83
84    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Access-
          Write">
85       <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
             role#Access"/>
86    </owl:Class>
87
88    <!-- http://ifi.uio.no/ontologies/health-role#Department -->
89
90    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#
          Department"/>
91
92    <!-- http://ifi.uio.no/ontologies/health-role#Department-Cardiology -->
93
94    <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#
          Department-Cardiology">
```

```
95      <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
            role#Department"/>
96    </owl:Class>
97
98    <!-- http://ifi.uio.no/ontologies/health-role#Hospital -->
99
100   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Hospital"
          />
101
102   <!-- http://ifi.uio.no/ontologies/health-role#Role -->
103
104   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Role"/>
105
106   <!-- http://ifi.uio.no/ontologies/health-role#Role-Doctor -->
107
108   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Role-
          Doctor">
109     <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
            role#Role"/>
110   </owl:Class>
111
112   <!-- http://ifi.uio.no/ontologies/health-role#Role-Nurse -->
113
114   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Role-
          Nurse">
115     <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
            role#Role"/>
116   </owl:Class>
117
118   <!-- http://ifi.uio.no/ontologies/health-role#Role-Patient -->
119
120   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Role-
          Patient">
121     <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
            role#Role"/>
122   </owl:Class>
123
124   <!-- http://ifi.uio.no/ontologies/health-role#Staff -->
125
126   <owl:Class rdf:about="http://ifi.uio.no/ontologies/health-role#Staff">
127     <rdfs:subClassOf rdf:resource="http://ifi.uio.no/ontologies/health-
            role#Hospital"/>
128   </owl:Class>
129
130   <!--
131   ///////////////////////////////////////////////////////////////////////////////////

132   //
133   // Individuals
134   //
135   ///////////////////////////////////////////////////////////////////////////////////

136    -->
137
```

```
138   <!-- http://ifi.uio.no/ontologies/health-role#Access -->
139
140   <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
          #Access">
141      <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Access-Read"/>
142      <accessTypeName>read</accessTypeName>
143   </owl:NamedIndividual>
144
145   <!-- http://ifi.uio.no/ontologies/health-role#Cardiologist -->
146
147   <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
          #Cardiologist">
148      <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Staff"/>
149      <hasRole rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Doctor"/>
150      <hasRole rdf:resource="http://ifi.uio.no/ontologies/health-role#Lege"
             />
151      <hasRole rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Physician"/>
152      <worksAt rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Cardiology"/>
153   </owl:NamedIndividual>
154
155   <!-- http://ifi.uio.no/ontologies/health-role#Cardiology -->
156
157   <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
          #Cardiology">
158      <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Department-Cardiology"/>
159   </owl:NamedIndividual>
160
161   <!-- http://ifi.uio.no/ontologies/health-role#Change -->
162
163   <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
          #Change">
164      <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
             Access-Write"/>
165      <accessTypeName rdf:datatype="http://www.w3.org/2001/XMLSchema#string
             ">write</accessTypeName>
166   </owl:NamedIndividual>
167
168   <!-- http://ifi.uio.no/ontologies/health-role#Doctor -->
169
170   <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
          #Doctor">
171      <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#Role-
             Doctor"/>
172      <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
             #Access"/>
173      <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
             #Change"/>
174      <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
```

```
        #Read"/>
175     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Review"/>
176     <roleName>doctor</roleName>
177 </owl:NamedIndividual>
178
179 <!-- http://ifi.uio.no/ontologies/health-role#Lege -->
180
181 <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
        #Lege">
182     <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#Role-
            Doctor"/>
183     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Access"/>
184     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Read"/>
185     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Review"/>
186     <roleName>doctor</roleName>
187 </owl:NamedIndividual>
188
189 <!-- http://ifi.uio.no/ontologies/health-role#Nurse -->
190
191 <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
        #Nurse">
192     <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#Role-
            Nurse"/>
193 </owl:NamedIndividual>
194
195 <!-- http://ifi.uio.no/ontologies/health-role#Patient -->
196
197 <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
        #Patient">
198     <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#Role-
            Patient"/>
199 </owl:NamedIndividual>
200
201 <!-- http://ifi.uio.no/ontologies/health-role#Physician -->
202
203 <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
        #Physician">
204     <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#Role-
            Doctor"/>
205     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Access"/>
206     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Read"/>
207     <hasAccessType rdf:resource="http://ifi.uio.no/ontologies/health-role
            #Review"/>
208     <roleName>doctor</roleName>
209 </owl:NamedIndividual>
210
211 <!-- http://ifi.uio.no/ontologies/health-role#Read -->
212
```

```
213     <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
            #Read">
214         <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
                Access-Read"/>
215         <accessTypeName>read</accessTypeName>
216     </owl:NamedIndividual>
217
218     <!-- http://ifi.uio.no/ontologies/health-role#Review -->
219
220     <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
            #Review">
221         <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
                Access-Read"/>
222         <accessTypeName>read</accessTypeName>
223     </owl:NamedIndividual>
224
225     <!-- http://ifi.uio.no/ontologies/health-role#Write -->
226
227     <owl:NamedIndividual rdf:about="http://ifi.uio.no/ontologies/health-role
            #Write">
228         <rdf:type rdf:resource="http://ifi.uio.no/ontologies/health-role#
                Access-Write"/>
229         <accessTypeName>write</accessTypeName>
230     </owl:NamedIndividual>
231 </rdf:RDF>
232
233     <!-- Generated by the OWL API (version 4.5.9.2019-02-01T07:24:44Z) https
            ://github.com/owlcs/owlapi -->
```

Listing 3.2: An Ontology

3.3.2.2 Semantic reasoner

The semantic reasoner and Rest API tool are developed to retrieve certain knowledge from the designed ontology and expose it to external requesters. The tool is designed as a web application so that it can expose different REST API endpoints to open up communication channel which accepts HTTP requests. It is implemented in Java Spring boot and uses OWLApi and Hermit Reasoner tools to help parsing the OWL file. The tool is packaged as a Web Application aRchive (WAR) file and deployed on the same tomcat server where ABAC engine runs on. Figure 3.4 illustrates the class diagram of the semantic reasoner tool. Spring boot application loads from ServletInitializer class that loads the application context and instantiates beans. DocumentController is the controller class that opens up APIs and routes request to required object class then returns an HTTP response. OWLDecision contains the complete business logic that parses OWL file and retrieves a 'Document' object which contains the synonyms. Below three endpoints are developed as part of thesis project:

- /query-by-role/{roleName}

- /query-by-access-type/{accessTypeName}
- /query-by-access-type/{accessTypeName}/for/{roleName}

'/query-by-role/{roleName}' endpoint accepts HTTP GET operation and receives role name as a parameter. The tools parse OWL file to find if any synonym for given role exists in the ontology. The ontology contains certain individuals for test purposes. For instance, if a request arrives at the endpoint with roleName = 'Lege', the response will be a list of synonyms including itself: ['Lege', 'Doctor', 'Physician'] as all of those individuals are belonging to the same class type Role-Doctor.

'/query-by-access-type/{accessTypeName}' endpoint is quite similar to '/query-by-role' with a slight difference of searching synonyms for given access type. Similarly, the ontology contains certain individuals for test purposes. For instance, if a request arrives at the endpoint with accessType = 'Review', the response will be the list of synonyms including itself: ['Review', 'Read', 'Access'] as all of those individuals are belonging to same class type Access-Read.

'/query-by-access-type/{accessTypeName}/for/{roleName}' has further filters over '/query-by-access-type' where it searches access type synonyms for given subject role. A query with accessType = 'Write' for roleName = 'Nurse' will return empty as such role is not given to nurses. Same endpoint will return ['Write','Change'] response for accessType='Write' for roleName = 'Lege'

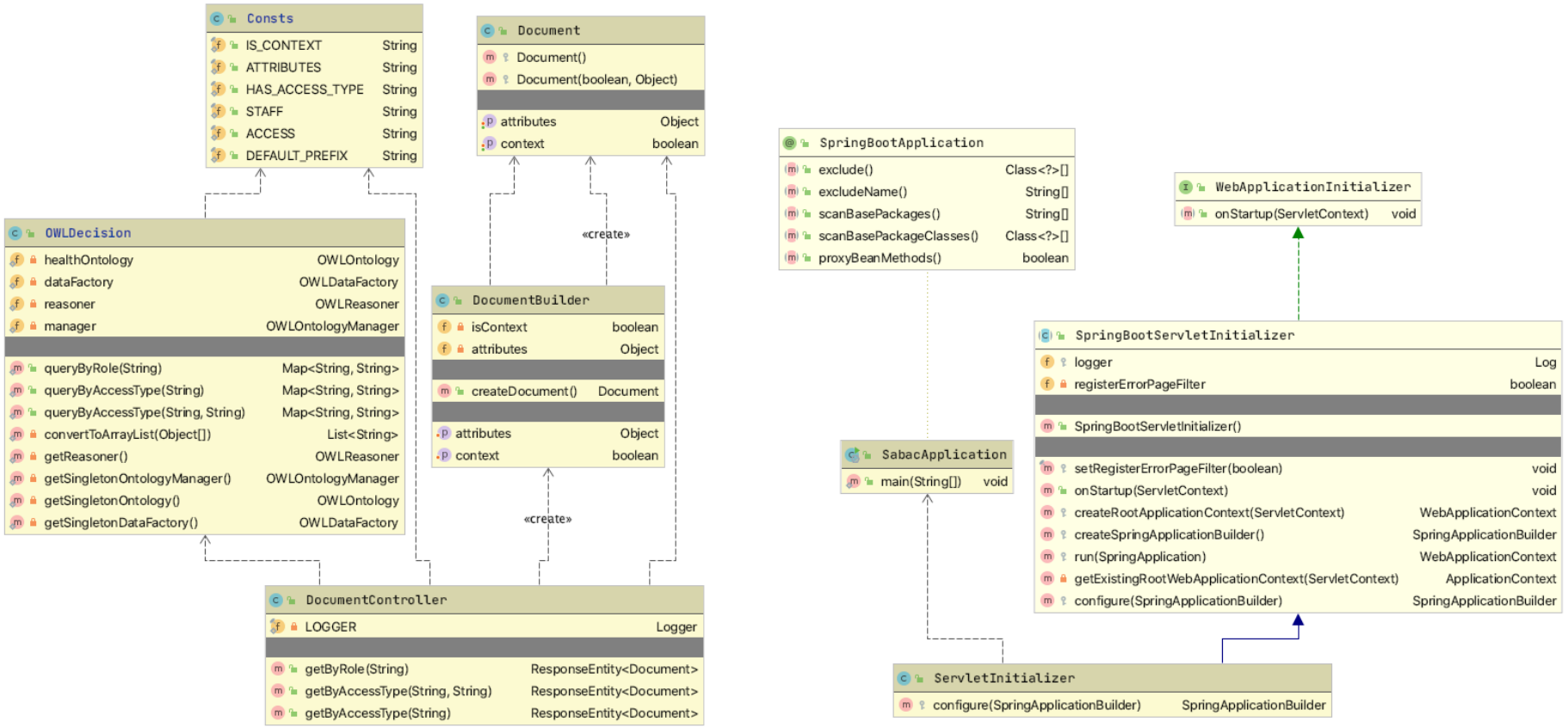

Figure 3.4: The semantic reasoner class diagram

3.3.2.3 Request Preprocessor

This is a PDP extension made possible by Authzforce team to write plugins on the ABAC engine. It is mainly provided for non-core (not defined in XACML 3.0 Core standard) PDP behavior and features to be implemented particularly to support specific XACML Profiles. In other words, new ways of processing XACML Requests before evaluation by the ABAC engine. Authzforce team has provided several other useful PDP extensions with different capabilities for different needs. However, Request Preprocessor is the ideal one for our need of integrating ABAC engine with semantic reasoner. This extension allows us to intercept the request – even modify it if needed – before evaluated by the PDP.

The plugin is a standalone java application packaged as a jar file and placed in already deployed ABAC engine library folder. In order to develop such a plugin, it is required to use a maven dependency or exported jar file for 'org.ow2.authzforce/authzforce-ce-core-pdp-api' provided by Authzforce team. The plugin requires a custom identifier that represents the extension and contains a set of business logic to manipulate or refactor the request. Once the tool is built, it should be placed in the ABAC engine lib folder and enabled with selected custom identifier.

The tool for the thesis project focuses on two attribute category ids, namely 'urn:oasis:names:tc:xacml:1.0:subject-category:access-subject' and 'urn:oasis:names:tc:xacml:3.0:attribute-category:action'. Once the plugin is enabled, any request that arrives at PDP is checked if contains any of those category ids. If any attribute is detected, then the plugin intercepts the request and communicates with the semantic reasoner to find any synonyms for the intercepted attributes. Upon fetching the synonyms, the request is refactored by enriching it with the synonyms. Listing 3.3 is a request arrived at the PDP and Listing 3.4 is the same request refactored by the plugin and forwarded to the PDP for the evaluation.

```
1 <Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xacml
    ="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance" xmlns:xf="http://www.w3.org/2005/xpath-
    functions" xmlns:md="http:www.med.custom.com/schemas/record.xsd"
    CombinedDecision="false" ReturnPolicyIdList="false">
2   <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:
      access-subject">
3     <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
        role" IncludeInResult="false">
4       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          doctor</AttributeValue>
5     </Attribute>
6   </Attributes>
7   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
      resource">
```

```
8     <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
          IncludeInResult="false">
9       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            doorLock</AttributeValue>
10    </Attribute>
11   </Attributes>
12   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
         environment" >
13    <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
          status" IncludeInResult="false">
14      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            emergency</AttributeValue>
15    </Attribute>
16   </Attributes>
17   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
         action">
18    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          IncludeInResult="false">
19      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            read</AttributeValue>
20    </Attribute>
21   </Attributes>
22 </Request>
```

Listing 3.3: XACML Request

```
1 <Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xacml
      ="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://www.
      w3.org/2001/XMLSchema-instance" xmlns:xf="http://www.w3.org/2005/xpath-
      functions" xmlns:md="http:www.med.custom.com/schemas/record.xsd"
      CombinedDecision="false" ReturnPolicyIdList="false">
2   <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:
        access-subject">
3     <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
          role" IncludeInResult="false">
4       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            doctor</AttributeValue>
5     </Attribute>
6     <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
          role" IncludeInResult="false">
7       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            lege</AttributeValue>
8     </Attribute>
9     <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
          role" IncludeInResult="false">
10      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            physician</AttributeValue>
11    </Attribute>
12  </Attributes>
13  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
        resource">
14    <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
          IncludeInResult="false">
```

```
15        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              doorLock</AttributeValue>
16      </Attribute>
17    </Attributes>
18    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
          environment" >
19      <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:custom:attribute:
            status" IncludeInResult="false">
20        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              emergency</AttributeValue>
21      </Attribute>
22    </Attributes>
23    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
          action">
24      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            IncludeInResult="false">
25        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              read</AttributeValue>
26      </Attribute>
27      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            IncludeInResult="false">
28        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              review</AttributeValue>
29      </Attribute>
30      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            IncludeInResult="false">
31        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              access</AttributeValue>
32      </Attribute>
33    </Attributes>
34  </Request>
```

Listing 3.4: XACML Request modified by PDP extension tool

The main purpose of this practice is to prove the integration between ABAC engine and semantic reasoner, as well as, take benefit of semantic reasoner to retrieve attribute synonyms and enrich the policy request to increase the chance of getting a valid permit on concerned object.

3.4 Evaluation

When the thesis subject was defined, it was also agreed to use Authzforce ABAC engine. It is known as one of most mature, open-source ABAC system with convenient level of community support behind. As it was the main component that other small integrations would depend on, the hardware requirement for it was dominant. Therefore, selected devices that will be used as residential gateways must meet at least the minimum hardware requirements to provide a satisfactory setup. Selecting the ideal device was set as RQ#3 in Section 1.3. When trying to find the ideal device, there were certain criteria kept in consideration. Although the computation power and embedded memory

capacity were important, the community support, Unix operating system, and platform capability of customizing for different programming languages were essential to be considered. The RPi Model 4B came along and appeared to be the most ideal device to select with its hardware, software capabilities, developer-friendly ecosystem, and community supports.

One other important area to discover was the importance of granularity in an access control system. As mentioned in Section 2.2, ABAC is a successor to other trend access control models with being a fine-grained access control system. In Section 2.2.1 we tried to explain the fine-graininess of ABAC with a real-world scenario in an e-Health domain, described what advantages it brings to an access control system, and finally address RQ#1.

Previous chapters describe the prototype implementation in detail. By installing and running the ABAC engine on a RPi device and connecting it to a local area network so that in-house IoT devices could get access through it, we achieve to decouple dependency between the internal IoT devices being connected to the external cloud environment. With that, we achieve to decentralize the access control system as all the in-house IoT devices get managed by an access control system in the fog-area. This prototype becomes a gate-keeper and controls the full authorizations of smart devices in the local area network. As the RPi devices can have a firewall as well as external IP addresses, it can receive requests from third party systems outside of the local network as well. In our scenario, a doctor working in a hospital may send a request to get authorization on a smart door lock of the patient's house when an emergency situation occurs. The permitted doctor can then open the door within emergency state with authorized batch card or biometric identity which can be a finger-print. As soon as the emergency state disappears, access on the door-lock automatically gets revoked. Overall prototype implementation is the achievement of RQ#4 in Section 1.3

3.4.1 Performance

Theoretically, the prototype setup containing S-ABAC allows us to achieve our goal of decentralizing the access control system. Carrying out the access control system from cloud to fog-area addresses the RQ#2. Furthermore, the S-ABAC setup provides the opportunity to strengthen the security level with a context-aware ABAC system thanks to the semantic reasoner and the ontology on top. Main concern on carrying such heavyweight setup from a cloud-area to a fog-area is no doubt that limited resource capacity. Provided solution must accommodate user satisfactory so that we can call it a successful project. Taking this into account, we created two environments, one on the cloud and another on a RPi, and performed load tests to indicate if our setup with underlying resource capacity would meet the same or similar performance. The load tests were performed with Apache JMeter tool by applying 100 concurrent

requests, each with different threads. The same tests were performed repeatedly on a plain ABAC system and an S-ABAC system on the cloud and fog areas. Table 3.1 indicates average response times in each tests. As listed in Table 3.1, there is a slight increase of 16.69 milliseconds between the plain ABAC setups, while it is 175.46 milliseconds between S-ABAC setups on the cloud and RPi. From user-experience point of view, they are both ignorable differences that we think it will not cause any inconveniences. On the other hand, performance differences between plain ABAC and S-ABAC is more visible on the cloud and RPi setups as both are more than 1.5 seconds. There are two main causes for these response time differences. First one is the additional network call from the ABAC to the semantic reasoner. The second one is that the semantic reasoner parses and retrieves the synonyms from an ontology for each and every request.

Table 3.1: Load test on cloud and RPi environments

Load Tests (ms)			
Cloud		RPi-4B	
ABAC	S-ABAC	ABAC	S-ABAC
52.36	1537.68	69.05	1713.14

3.4.2 Project Cost

The prototype setup consists of a RPi Model 4B device and all other software applications running on it. The RPi Model 4B offers a Micro-HDMI display port to connect a screen. For that, a Micro-HDMI to HDMI adaptor was purchased. Besides that, personal equipment such as laptop, screen were used. For the software implementation, Intellij IDEA with free student account, Postman, JMeter, Authzforce ABAC software, and Protégé Editor tools were used. There was not a license cost for any of these tools. As the project was designed as a prototype, no any smart equipment was used. Instead, Postman was used as a simulator to send mock requests. Table 3.2 presents prototype costs.

Table 3.2: Prototype equipment costs

Equipment	Cost
RPi Model 4B	729 Kr
Mini-HDMI Adapter	95 Kr
Total	824 Kr

3.4.3 Limitation

Minimum hardware requirements for the ABAC engine were shared in earlier chapters. In the project implementation, the hospital staff ontology created

for thesis project meets just a prototype prerequisite. However, in a real-world scenario, an enhanced ontology might be required. Similarly, ABAC engines that were installed on the cloud and RPi were hosting a single domain and a single policy set. Performance of the semantic reasoner tool that parses the ontology might differ on a more complex ontology. Although current storage capacity would suffice, the response time might get longer than the provided average load test results. On the other hand, the RPi Model 4B offers a larger storage capacity version that comes with 8GB RAM. If further integrations are required, this version can be preferred.

CHAPTER 4

Discussion and Future Work

4.1 Critical Discussion

The OASIS XACML standard contains many useful functions that help ABAC become a fine-grained access control system. The individuals' needs may vary and existing functions may not be enough to establish certain goals. For such cases, Authzforce provides an extension [23] where individuals can create their own functions that do not exist in the XACML standard. In this thesis, it is preferred to enrich an XACML request by appending synonyms that are fetched from the ontology and then perform 'urn:oasis:names:tc:xacml:1.0:function:string-equal' to compare attribute values. Being able to modify the XACML request on the PDP level gives extensive control to the user. However, performance-wise a more efficient solution only for this particular use-case is to create a new function named 'string-equal-or-synonym' where it will first check if two attribute values do match. If they do not match, then it will call semantic reasoner to obtain synonyms and repeat the string-equal operation for them as well.

On one hand, ABAC offers a fine-grained access control system and it is known as being more flexible and more secure compared to other access control systems described in Chapter 2. On the other hand, as a secure and fine-grained access control system, it accommodates complexity both in terms of implementation as well as computation. Therefore, the common practice for IoT service providers is running the ABAC system either on a cloud or in a distance from client's location and maintain the communication over encrypted network calls. Hence, the underlying resources can be utilized as best as possible. This is a typical centralized managed service model where all decisions are made by one single service for all clients.

Centralized service model brings certain benefits to the service providers such as reducing the cost and quick implementation and deployment. No doubt that the biggest risk in centralized model is to become a single point of failure. That is even more risky for such critical services like an access control system.

Keeping in mind that the cloud infrastructure can scale up and down with elasticity which may reduce the risk of failure under interchangeable workload. Nevertheless, providing such a fail-safe service design with redundancy in place increases the Service Level Agreement (SLA) cost which then impacts service cost especially on public cloud.

Running an S-ABAC on the cloud infrastructure reveals another concern, which is General Data Protection Regulation (GDPR). EU law enforcement requires citizens related data to be stored in the EU; so, it is subject to the European privacy law. In the cloud technology, the underlying infrastructure can be in any data-center in the world if the right region and zone are not selected. Furthermore, GDPR requires organizations to be able to give individuals their data in a usable format and even delete it if requested by the owner. This should also apply to the backups. However, cloud technologies are not as flexible as standalone servers to apply such requirements right away. Hence, storing sensitive information such as access policies, ontologies and individual data may not be compliant with the GDPR in some cases where anonymization process is not applied.

Running the S-ABAC on a residential gateway in a fog-area turns the system into a decentralized setting. Main advantage of this model is having complete control. Only authorized users may send requests from outside of the fog-area. Whereas internal communications would work faster and secure with being isolated from the outside world and protected with a firewall. However, IP address whitelisting for external requesters is a good practice to improve the security level for such applications running on limited resources. Another benefit is, in case of a failure in the system, that it does not impact other individuals but itself. Number of individuals being impacted from a failure may reduce from thousands to one. Finally, decentralized model enables less external but more internal network communications. This model also supports reducing network overburden globally.

4.2 Future Work

4.2.1 Reading Synonyms From a Repository

Individuals are the basic instances of objects, in other words, "ground level" components of an ontology. The individuals in an ontology may represent concrete objects such as humans, animals, and tables plants. Strictly speaking, an ontology does not need to include an individual. However, the general purpose of an ontology is to provide a means of classifying individuals. In our ontology, we used static individuals to represent subject-role and access-type synonyms, which were just enough to prove a concept. However, in a real-world scenario, the list of synonyms might be larger or might have potential to increase. In such cases, a better practice would be using a repository which

would hold the list of individuals so called synonyms and semantic reasoner would map repository entities with ontology as and when needed. This will keep the OWL clean and avoid it to get larger in size.

4.2.2 Response Postprocessor Extension

Authzforce team provides very useful extensions to support improving the ABAC engine's capability for different needs. In this project, the 'Request Preprocessor Extension' [24] is used for intercepting the request before being evaluated on the PDP. Incoming requests get enriched with appending attribute synonyms from the semantic reasoner to increase the chance of getting a 'Permit'. Nevertheless, requests may not get 'Permit' with the appended synonyms, even may get 'Indeterminate' response if something goes wrong. In order to collect more statistics on responses or maybe to take some actions for different response types, another useful extension, i.e., 'Response Postprocessor Extension' [25], can be used. Similar to the 'Request Preprocessor', this extension pauses the flow and allows a user to automize some actions. However, in contrast to the 'Request Preprocessor', this extension intercepts the response before it is delivered to the user. Creating such an extension can be useful to record the responses and run different analytics on them. Hence, the users can have an overview of the requests and responses.

CHAPTER 5

Conclusion

During the initial discussion of the thesis we set four main goals to achieve throughout the semester. These goals were also referring to the research questions mentioned in Section 1.3.

The **first** goal was to discover the importance of ABAC system compared to other trend access control systems, especially in e-Health domain. We also wanted to further investigate the fine-graininess of an access control model and the advantages that it offers in IoT. The background study along with literature review aimed to answer the RQ#1 and were covered in Chapter 2.

The **second** goal was to study the feasibility of decentralizing a heavyweight access control system, ABAC. Main purpose of decentralization was to carry the access control system to fog-area, where end-user has complete control over authentication and authorization. Furthermore, this would avoid the centralized approach's bottleneck, being a single point of failure. In order to achieve this goal, we needed to set the **third** goal to do a resource assessment and identify the ideal device to use in our prototype. In Chapter 2 we analyzed different IoT gateways and in Chapter 3 we discovered the ABAC engine's minimum hardware requirements. With these analyzes, we decided RPi Model 4B is the best match for our needs. The review and assessments in above-mentioned chapters addressed RQ#2 and RQ#3.

The **fourth** and the final goal was to extend the capability of ABAC system by integrating it with a semantic reasoner and offer S-ABAC solution on fog-area, running on a resource constrained residential gateway. The achievement of this goal was separated in a few different categories. Initially, we needed to gain knowledge on semantic technologies and ontologies in IoT. Chapter 2 contains the discussions on those two topics. Next, we identified the need of creating a hospital staff ontology by using Protégé app and parse it in a semantic reasoner tool that would accept requests over the internal network and answer with the extracted knowledge from ontology. Later, we needed to identify an integration point where we can link the ABAC engine to the semantic reasoner

tool. In order to discover that, we analyzed ABAC engine source codes on GitHub [26], Authzforce developer guide [24] and other technical forums. Finally, we developed a plugin that intercepts request on ABAC before it is evaluated on PDP and manipulate the request based on the synonyms retrieved from semantic reasoner. Chapter 3 contains the complete implementation in detail, and the evaluation of the S-ABAC. The Section 3.4 that contains the evaluation discussion proved RQ#4 is achieved.

As a conclusion, the thesis presented a prototype implementation for Semantic - Attribute-Based Access Control system on a residential gateway running on lightweight micro-computer Raspberry Pi Model 4B. As a fine-grained access control system, ABAC, with support of an ontology and semantic reasoner, proved to strengthen its capabilities and become more resilient with the use of semantic relationships while inferring implicit policies. Implementation proved that moving such a heavyweight and considerably complex system from cloud platform to fog area did not have any computational resource constraint, yet brought more control to the fog area. Providing secure communication channels over https-enabled the IoT gateway receive request from an external system, such as a hospital management system, to grant permit on an IoT device in smart home care.

The analyses and implemented solution showed that by providing an ontology-based context-aware solution, S-ABAC is a successor of other access control systems. Therefore, it is ideal for hosting highly sensitive and private information. Having such an enhanced system on fog area also makes the solution GDPR compliant. With further integration and improvement suggested in "Future Work", designed solution can become a commercial product and be used in many industries, primarily in healthcare.

Bibliography

[1] Tracy Cozzens. *Terra Drones fight coronavirus with delivery drones*. Mar. 2020. URL: https://www.gpsworld.com/china-fights-coronavirus-with-delivery-drones/ (visited on 26/04/2020).

[2] Helsenorge. *Smittestopp app*. Apr. 2020. URL: https://helsenorge.no/coronavirus/smittestopp (visited on 26/04/2020).

[3] Symantec. *Internet Security Threat Report 2018*. Mar. 2018. URL: https://docs.broadcom.com/doc/istr-23-2018-en (visited on 05/05/2020).

[4] Liyanage, Madhusanka. *IoT security : advances in authentication*. eng. Hoboken, New Jersey, 2020.

[5] Andaloussi, Y. et al. 'Access control in IoT environments: Feasible scenarios'. In: *Procedia Computer Science* vol. 130 (2018), pp. 1031–1036. DOI: 10.1016/j.procs.2018.04.144. URL: https://doi.org/10.1016/j.procs.2018.04.144.

[6] Aftab, M. u. et al. 'The Evaluation and Comparative Analysis of Role Based Access Control and Attribute Based Access Control Model'. In: *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. 2018, pp. 35–39.

[7] Dodig-Crnkovic, Gordana. ''Scientific methods in computer science.' In: Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia'. In: (2002), pp. 126–130.

[8] Mohammed, Ennahbaoui and Said, El Hajji. 'SWOT Analysis of Access Control Models'. In: *International Journal of Security and Its Applications* vol. 8, no. 3 (Aug. 2014), pp. 407–424. URL: http://dx.doi.org/10.14257/ijsia.2014.8.3.08.

[9] Benantar, Messaoud. *Access Control Systems : Security, Identity Management and Trust Models*. IBM Corp.Austin USA: Springer, Boston, MA, 2006. ISBN: 9780387004457.

[10] Ausanka-Crues, Ryan and Mudd, Harvey S. 'Methods for Access Control : Advances and Limitations'. In: 2006.

[11] *RBAC 2000: Proceedings of the Fifth ACM Workshop on Role-Based Access Control*. Berlin, Germany: Association for Computing Machinery, 2000. ISBN: 158113259X.

[12] axiomatics. *The Evolution of RBAC Models to Next-Generation ABAC:* URL: https://informationsecurity.report/Axiomatic/The\%20Evolution\%20of\%20RBAC\%20Models\%20to\%20Next\%20Generation\%20ABAC.pdf (visited on 13/06/2020).

[13] Hu, Vincent C et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Jan. 2014. DOI: doi.org/10.6028/NIST.SP.800-162.

[14] Domingue, John, Fensel, Dieter and Hendler, James A., eds. *Handbook of Semantic Web Technologies*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-540-92913-0. URL: https://doi.org/10.1007/978-3-540-92913-0.

[15] Foundation, Raspberry Pi. *Raspberry Pi 4 Model B product brief*. May 2020. URL: https://static.raspberrypi.org/files/product-briefs/200521+Raspberry+Pi+4+Product+Brief.pdf.

[16] Poyen, C Eng Faruk. 'Raspberry Pi and its Use in IoT Applications'. In: Jan. 2019, pp. 42–47. ISBN: 978-81-938404-4-3.

[17] Hughes, John M., ed. *Arduino: A Technical Reference*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc, May 2016.

[18] Arduino. *About Arduino*. URL: https://www.arduino.cc/en/Main/AboutUs.

[19] Fiware. *Fiware About-us*. URL: https://www.fiware.org/about-us (visited on 23/06/2020).

[20] EU Commission. *FIWARE – a European success story*. Mar. 2017. URL: https://ec.europa.eu/digital-single-market/en/news/fiware-european-success-story (visited on 23/06/2020).

[21] Fiware Community. *Autzforce CE, ABAC engine*. URL: https://authzforce-ce-fiware.readthedocs.io/en/latest/InstallationAndAdministrationGuide.html#system-requirements (visited on 05/05/2020).

[22] Jason Fox. *ABAC Tutorial*. URL: https://fiware-tutorials.readthedocs.io/en/latest/administrating-xacml/ (visited on 23/06/2020).

[23] Cyril Dangerville. *PDP Extensions Tutorial - Function Extension*. URL: https://authzforce-ce-fiware.readthedocs.io/en/latest/UserAndProgrammersGuide.html#function-extensions (visited on 25/06/2020).

[24] Cyril Dangerville. *PDP Extensions Tutorial - Request Preprocessor*. URL: https://authzforce-ce-fiware.readthedocs.io/en/latest/UserAndProgrammersGuide.html#request-preprocessor-extensions (visited on 25/06/2020).

[25] Cyril Dangerville. *PDP Extensions Tutorial - Result Postprocessor*. URL: https://authzforce-ce-fiware.readthedocs.io/en/latest/UserAndProgrammersGuide.html#result-postprocessor-extensions (visited on 25/06/2020).

[26] Cyril Dangerville. *ABAC Core Source Code*. URL: https://github.com/authzforce/core (visited on 14/07/2020).