University of Aveiro Department of Electronics, Telecommunications and Informatics

2015.

Vedran Semenski

An ABAC framework utilizing XACML for IoT applications

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em (designação do Mestrado), realizada sob a orientação científica do Doutor (nome do orientador), Professor (categoria do orientador) do Departamento de (designação do departamento) da Universidade de Aveiro

texto Apoio financeiro do
(se aplicável)

texto Dedico este trabalho à minha esposa e filho pelo incansável apoio.

(opcional)

o júri

presidente	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor João Antunes da Silva professor associado da Faculdade de Engenharia da Universidade do Porto

agradecimentos

texto O presente trabalho propõe-se divulgar as mais significativas técnicas de construção existentes em Portugal continental. O livro é composto por uma apresentação dos materiais tradicionais de construção (suas principais características), uma compilação de fichas técnicas (de carácter prático, uma vasta bibliografia comentada e um glossário de termos técnicos.
A importante colaboração de diversas personalidades ligadas à área da História da Arquitectura, bem como o levantamento fotográfico realizado contribuem para o conhecimento e valorização de um saber tradicional.

(opcional)

palavras-chave

Access Control, ABAC, XACML, IoT, Big Data, NoSQL, XML, JSON, SMARTIE, Smart City, M2M, Security

resumo

IoT (Internet of Things) and Big Data are technologies are commonly interconnected as one is usually dependent on the other in one way or another. Developments in sensor and microcontroller technologies are one or the pushing factors that are responsible for showing the potential significance of the implementations therefore giving them significance. Implementation scenarios wary from smaller scale to larger scale and the more significant ones are Smart City applications that commonly include many sub-systems like smart traffic control, energy and utility consumption and production monitoring, smart building management etc. Others include smart houses, health personal health monitoring, etc. As these application more often than not use confidential and/or personal information, a security component is needed to protect from unwanted intrusion, stealing or misuse. OASIS (Organization for the Advancement of Structured Information Standards) developed the XACML (eXtensible Access Control Markup Language) standard for writing/defining requests and policies in either a XML or JSON format and the evaluating of the requests over sets of policies for the purpose of enforcing access control over resources. It is defined so the requests and policies are readable by humans but also have a well defined structure allowing for precise evaluation. It utilizes ABAC (Attribute Based Access Control) which compared to RBAC (Role Based Access Control) systems is more difficult to implement but offers greater flexibility, lower maintenance requirements because of the adaptive nature of ABAC, and because of that, an overall more secure system. This work aims to create a security framework that utilizes ABAC and the XACML standard so that it can be utilized in other systems and enforce access control over resources that need to be protected by allowing access only to authorised subjects. It will also allow for fine grained defining and evaluation for more precise and therefore providing a greater level of security. This solution will also require NoSQL databases for storing policies and storing sensor data as the primary scenario considered is the integration in a Smart City application.

keywords

Access Control, ABAC, XACML, IoT, Big Data, NoSQL, XML, JSON, SMARTIE, Smart City, M2M, Security

abstract

IoT (Internet of Things) is an area which offers great promise and although a lot of core problems already have satisfactory solutions, security has remained somewhat unaddressed and remains to be a big issue. Access Control is a security technique that evaluates requests for accessing resources and denies access if it is unauthorised therefore providing security for vulnerable resources. As Access Control is a broad term that consists of several methodologies of which the most significant are: IBAC (Identity Based Access Control), RBAC (Role Based Access Control) and ABAC (Attribute Based Access Control). In this work ABAC will be used as it offers the most flexibility compared to IBAC and RBAC and because of ABAC's adaptive nature it also offers lower maintenance requirements. OASIS (Organization for the Advancement of Structured Information Standards) developed the XACML (eXtensible Access Control Markup Language) standard for writing/defining requests and policies and the evaluation of the requests over sets of policies for the purpose of enforcing access control over resources. It is defined so the requests and policies are readable by humans but also have a well defined structure allowing for precise evaluation. The standard uses ABAC. This work aims to create a security framework that utilizes ABAC and the XACML standard so that it can be used in other systems and enforce access control over resources that need to be protected by allowing access only to authorised subjects. It also allows for fine grained defining or rules and requests for more precise evaluation and therefore a greater level of security. The primary use-case scenarios are large IoT applications such as Smart City applications including smart traffic monitoring, energy and utility consumption, personal healthcare monitoring and etc. These application deal with large quantities (Big Data) of confidential and/or personal data. A number of NoSQL (Not Only SQL) solutions exist for solving the problem of volume but security is still an issue. This work will use two NoSQL databases. A key-value database (Redis) for the storing of policies and a wide-column database (Cassandra) for storing sensor data and additional attribute data during testing.

Table of content

List of Acronyms

ABAC	Attribute-Based Access Control
ACID	Atomicity, Consistency, Isolation and Durability
API	Application Program Interface
BASE	Basically Available, Soft State and Eventually Consistent
BSON	Binary JSON
CQL	Cassandra Query Language / Cypher Query Language
DAC	Discretionary Access Control
DBMS	Database Management System
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDE	Integrated Development Environment
IBAC	Identity Based Access Control
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
M2M	Machine to machine
MAC	Mandatory Access Control
NFC	Near Field Communication
NoSQL	Not Only SQL
OASIS	Organization for the Advancement of Structured Information Standards
OLAP	Online Analytical Processing
OLTP	Online transaction processing
OWASP	Open Web Application Security Project
PAP	Policy Administration Point
PDP	Policy Decision Point

PEP	Policy Enforcement Point
PIP	Policy Information Point
PII	Personal Identifiable Information
PRP	Policy Retrieval Point
RBAC	Role Based Access Control
REST	Representational State Transfer
RDBMS	Relational Database Management System
RFID	Radio-Frequency Identification
SMARTIE	Smart City
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
W3C	World Wide Web Consortium
UDAF	User-Defined Aggregate Functions
UDF	User-Defined Functions
UDTF	User-Defined Table Functions
UWB	Ultra-Wide Band
XML	eXtensible Markup Language
XACML	eXtensible Access Control Markup Language
YARN	Yet Another Resource Negotiator

List of figures

List of tables

Nisu pronađeni unosi u tablici slika.

1 Introduction

Tendencies and advancements in sensor technology gave the birth to the Internet of Things and Big Data. These technologies offer great opportunity and applications that could and are changing everyday life, but also have many issues that could be exploited in a malicious manner. Gathering and using vast amounts of data/information pose many problems. Along with the initial and core implementation problems regarding, connection, communication, storage and processing which already have a number of advancements and usable developments, security of information is an issue that needs to be addressed in order to make the technology safe. Smart City projects are one of the biggest and most significant implementation scenarios and because of that must offer greater levels of security and need to implement long term and flexible solutions. Implementations include smart traffic control, energy consumption monitoring and management, health monitoring and others. Data is generated by sensor networks, people's smart phones, other data sources like healthcare, education and others. Therefore the information that has to be managed and used is often private or confidential. The control over actuators, sensors and data can be misused if access is given to unauthorised subjects. These problems therefore need to be addressed for the solutions to work in a safe and secure manner. Access control is a broad term and represents a way of securing/limiting access to resources which can be anything from services, actions or data so only authorised subjects have access. This dissertation will address that problem and propose a solution utilizing Attribute Based Access Control (ABAC), the OASIS XACML standard and data storage using NoSQL databases.

Section 1 is divided as follows: Section 1.1 contains the official short description text along with the objectives, Section 1.2 will present the problem addressed by this work, in Section 1.3 a short description of the proposed solution will be given and Section 1.4 will present the structure of this dissertation,

1.1 *Dissertation description text*

SMARTIE (http://www.smartie-project.eu/index.html) is an European project where PT – Inovação e Sistemas is one of the partners. The project is aimed at creating a framework (IoT) to collect and share large volumes of heterogeneous sensor information (Big Data) to be used in smartcities applications. In order to protect sensitive data, a security component is necessary to manage authentication and authorization which will be based on the XACML standard. The original XACML standard is based on XML but in SMARTIE we will try to use JSON.

Objectives:

This dissertation is focused on two main development activities: databases and security framework. It comprises the following tasks to be executed:

- Become familiar with SMARTIE;
- Become familiar with IoT;
- Become familiar with ABAC models and particularly with XACML;
- Become familiar with NoSQL databases (Big Data);

- Selection of the NoSQL database to be used to store the collected sensor data;
- Selection of the database to store the security policies to be enforced;
- Conceptual, logical and physical models for both databases;
- Security framework based on XACML.

All the work will be developed at Instituto de Telecomunicações facilities.

1.2 *Problem formulation/description*

SMARTIE is a Smart City project founded by the European Union and PT – Inovação e Sistemas is one of the partners contributing to the development of the project. The goal of the project is creating a framework for utilizing sensor networks, storing processing and managing of data and to provide a platform for creating Smart City applications. The notion of storage and processing of large amounts of data needs to be addressed with the utilisation of NoSQL databases and related systems. Other than that, security of data is the focus of this work. Because the data that will be used in these applications will often be personal or confidential, a security framework is needed to provide protection. Access control is one way of dealing with this issue and .Role Based or Identity Based Access Control is the way access control is commonly enforced. These are somewhat simple to build and the relation between roles allowed actions are intuitive to comprehend, a common issue is the explosion of the number of roles and the managing/maintaining of them. Also there is the issue of somewhat limited possibilities on defining conditions, actions on certain resources etc. The OASIS's XACML utilizing Attribute Based Access Control (ABAC) will be explored and needs to be utilized in the built security component. OASIS is a non-profit consortium produces open standards in the areas of security along with others. It defined the XACML standard for creating request, policies and their evaluation in the purpose of managing access to resources. Although the initial version of the standard was based on XML, SMARTIE will try to utilize the JSON variant therefore this dissertation should also focus on the JSON variant. ABAC offers many benefits over other types of access control and although the XACML v3.0 standard has been available since January 22., 2013., a complete and flexible solution enforcing ABAC and the XACML standard has still not been made open source or available in other ways.

This dissertation will explore the areas of IoT, M2M, Big Data, NoSQL, Access Control, XACML, JSON and SMARTIE. The goals can therefore be defined as becoming familiar in the areas mentioned before, utilizing a NoSQL solution for gathering sensor data, creation of a security framework for enforcing access control that utilizes the OASIS XACML standard and the accompanying NoSQL solution for storing policies. SMARTIE is an implementation scenario that needs to be considered in the development and testing of the solution but of course shouldn't be limited to that use-case.

1.3 *Proposed solution*

An ABAC framework utilizing the XACML standard offers many benefits to the user. The solution utilizes the architecture described in [1] and offers great flexibility depending on the use/implementation. The solution is built using open source components built [2] along with other open source NoSQL solutions. The implementation and final solution is explained in more detail later in the document. The enforcement of access control is done by implementing of an interface which is used for executing actions over resources and sending it together with the XACML request either in JSON or XML format. The solution returns the evaluation result and executes the action in the case of a positive result. The framework offers a simple and fast way of integrating access control in an existing application by providing 2 main entry points. An administration point that is used for managing policies and an enforcement point that needs to be called at the point where the user wants to enforce access control. Because the system uses ABAC and XACML it is very flexible and allows for having complex enforcement methods by defining policies around the attributes of the subject, resource and environment. Compared to RBAC systems this therefore needs less management because there is no need for updating roles after certain changes happen and as a result is less prone to errors over time. The critical part from an implementation perspective is having well and correctly defined policies and requests.

1.4 *Dissertation structure*

The structure of this dissertation is organized as follows: In Section 2 present the current State of the Art along with critical overview and presenting some related work, Section 3 contains the description of the proposed and developed solution, Section 4 presents test results in various test scenario, Section 5 presents an integration proposal for SMARTIE and test results, Section 6 contains the final conclusion and presents potential future work and the last Section contains the list of references used in this dissertation.

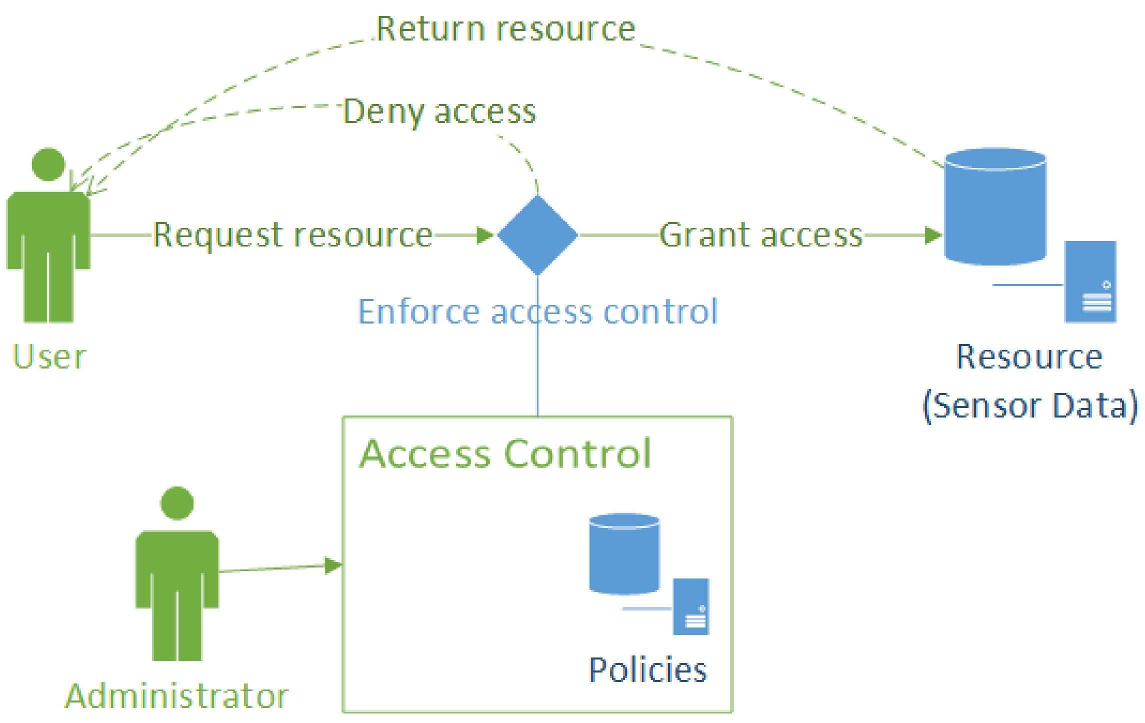


Figure 1. High - level overview of the architecture

A high - level overview diagram of the proposed solution can be seen in Figure 1. It shows how the solution will be integrated to enforce access control and how will it be used by users/system administrators.

The solution allows for 2 ways of integrating into other systems in need of access control. The first possibility is putting and running the solution on the same place where the enforcement of access control needs to be. This offers greater security, performance, configuration options and overall control. On the other hand it needs to be configured and at a minimum, it needs to provide a NoSQL database (Redis) for storing policies. The other option would be using the solution as a REST service and communicating with it trough a secured connection. These options allow the system using this service to use and configure the solution as needed. The detail of the solution will be explained in greater detail in later sections in this document.

2 State of the art

This section contains short overviews of areas related to the work done in this dissertation. The overviews contain brief descriptions, analysis of the current state and work related to those areas along with a future oriented perspective.

2.1 *The Internet of Things (IoT)*

In this section the current state of IoT will be presented. The main areas that will be briefly presented and analyzed are: current state and overview, general concerns, recent work and studies, technologies used, implementation examples and expected future developments.

The IoT is a recent paradigm in the area of networks and communication that has had a lot of growth and new developments in recent years. Although there are a lot of definitions of the IoT and the somewhat changing and branching nature of the development trends in this area the basic and simplified idea of this concept is that nowadays everyday objects can be equipped with cheap microcontrollers, sensors, means of connecting to one another and the Internet. The devices can generally be divided into two categories: sensors and actuators, meaning that their purpose is to provide and share data or some kind of readings or to receive commands and react/complete actions accordingly. The overall implementation and use of this could be used in a number of applications including: home automation, industrial automation, medical aids, mobile health care, elderly assistance, intelligent energy management and smart grids, automotive, traffic management and many others [1] [2]. All of these offer beneficial and significant impact on almost all areas of everyday life and have potential for providing advancements in research areas unrelated to networking or computer science. The definition of IoT is not exact as the authors of [1] wrote. They explain that the two main views come from the name "Internet of Things". One vision views the concept from a network perspective concentrating on communication and connection problems while the other is oriented on the "Things" or object perspective concentrating on sensor technologies, new communication technologies like RFID and NFC ,and integration into other devices in a seamless and affordable way. A third view is also present which the authors defined as a "Semantic oriented vision" which is concentrated more on the use, implementation and processing of data.

A number of challenges are in the way of successfully building and utilizing the IoT. Scalability is one of the obvious challenges. Any kind of IoT application that requires a large number of devices commonly face problems with response time, memory, processing and energy constraints. Other challenges include security issues. The data being generated by sensor networks is huge and could over saturate the network, the data could be personal and as such has to be protected from unwanted access. Attacks by hackers, malicious software, viruses and other sources can also disturb the flow and integrity of data/information. As the authors of [3] [4] [5] describe in their work and stress the importance of security for a widely acceptable solution. In their work they describes the IoT divided into 3 or 4 layers and define security requirements for every layer providing the current state of technologies used in these areas. They stress the need for an uniformed and standardised open architecture and solution. Solutions for many of these problems are already conceptually known and are in some examples implemented but the lack of a open and standardized solution is certainly an issue that would proved to be beneficial if/when solved.

Technological advancements in various sensor modules and cheap and energy efficient microcontrollers along with recent communication technologies like RFID, NFC and network protocols are the main factors that enabled the fast development and spreading the IoT. Because of these advancements the concept became a reality and is gaining importance and more uses and applications. As the means of connection and communication are open and utilizes a number of older and newer technologies the networks and number of devices is fast growing and also brings up the problem of standardisation and implementation of standards in the purpose of uniformly solving known problems with for instance security, integrity and scalability.

A more future oriented view and analysis is provided by [6]. It describes a more human centric rather than thing centric future of the IoT. Devices being linked to people and monitoring their state and condition. Others are used for monitoring or controlling things in the environment but always in close relation to human needs and/or wants. It describes a need for IoT applications to provide better quality of service and seamless integration into areas of life. The more relevant and faster growing application domains are: Environmental Monitoring, Smart Retail, Smart Agriculture, Smart Energy and Power Grids and Smart Healthcare. The architecture proposed is separated into 5 layers stacked one on top another in this order:

1. Things - containing devices or elements that are data generators and/or consumers of information. The devices could range from small and unintelligent embedded systems to complex devices or virtual entities. The communication would be done different communication technologies and a wide variety of protocols.
2. Network - this layer contains functionality and means of managing a sensor network. Discovery of new devices, maintenance, scalability, universal abstractions of the Inputs and Outputs and general abstraction for the upper and lower layers and enabling plug-n-play like use using already known models like push/pull or publish/subscribe models and REST based protocols.
3. Data Management - it is defined as "Big-Little" Data Management referring to the data usually generated by sensor networks meaning that the data generated by for instance temperature sensors is small in individual reading size large considering a large number of sensors over a period of time. This layer is responsible for categorizing and aggregating data retrieved from the Network layer in order for it to be used and/or by the Analytics layer. Data generated by sensors is often slow changing so this fact should be taken advantage of for more efficient data storage.
4. Analytics - this layer mines/retrieves data from the Data Management layer and performs data processing and analysis depending on the type of data and end user of the result. Is provides the applications (which would be located on top) with useful data and information for subsequent use.

There a lot of commercial implementations of IoT on a smaller scale which are mainly focused on personal use in home energy consumption, health monitoring and environment monitoring applications. These solution usually utilize custom solutions along with custom hardware (regarding the sensor devices).

Introducing standards that would be accepted and the creation of publicly available frameworks that utilize those standards would be very beneficial and would allow easier further developments. The lack of said standards and frameworks mean during initial developments of applications many of them face the same problems and implement a custom solution. This leads to incompatibility issues and stand in the way of a truly global IoT. The surveys and proposals given in [5] [3] give a good overview of the current situation in IoT and a give proposals for more uniformed future research and developments.

2.2 *Machine to Machine communication (M2M)*

This section will provide a brief overview of the current state, issues and a future developments in M2M. M2M, which means "Machine to machine" refers to the technologies used for communication between devices. It is a broad term that can sometimes be used for terms like "Machine to Mobile" and "Man to Machine" which mainly refer to the implementation in a more precise manner. It is also commonly regarded in the context of IoT because it is essential part if IoT. IoT concentrates more on a higher overview considering network problems and viewing object in a more general way, while M2M deals more with one-to-one communication between devices and the functioning of devices regarding one another, less regarding the overview from a network perspective [7]. The motivation behind M2M comes as stated in [8] mainly comes from two observations:

1) a networked machine is more valuable than an isolated one
2) when multiple machines are effectively interconnected, more autonomous and intelligent applications can be generated
3) smart and ubiquitous services can be enabled by machine-type devices intelligently communicating with other devices at anytime and anywhere

M2M systems are mainly used in the areas of personal health monitoring and smart homes/smart houses. M2M communication can be achieved with a number of technologies but the most significant and most promising ones are wireless technologies. In [8] the main technologies that are mentioned and review are: Zigbee, Bluetooth, UWB, IEEE 802.15.6, Wi-Fi, HomeRF, 60GHz transmission and Visible light communications. They offer easy integration of devices and many options for developing application. Applications utilizing M2M systems are usually focused on improving the quality of life for individuals. The two areas that are most significant and are the main drivers for developments in M2M are "personal health monitoring" and "smart home" applications. Trends in M2M as mentioned in [8] suggest exponential growth in the number of M2M-enabled devices. From 50 million in 2008 and over 200 million in 2014, it is expected to grow up to 50 billion in 2020. Along with communication and integration challenges, security is one of the more important areas that need to be deled with regarding M2M.

The architecture of M2M systems can generally be divided into 3 layers.

1) Terminal layer - contains the access gateway , area network and M2M nodes, M2M enabled devices
2) Network layer - responsible for transmitting data between the other two layers
3) Application layer - contains the application that utilizes the M2M system

Security issues in M2M systems are similar to ones found in providing security and privacy in communication over other networks, the internet... Additional problems that are more specific to M2M occur in the Terminal layer. Devices could potentially be easily accessible to attackers. This means that they could be tampered with or be controlled by attackers and false data could be injected threatening the overall performance of the M2M system and application.

M2M systems in the IoT context provide the means to build applications that can further the developments in many scientific areas along with bettering personal quality of life. Solving security issues and the general standardisation of undefined aspects in communication, as well as the development of quality and publicly availably solutions are the future steps that will enable the wide commercial use of these technologies.

Work by the authors of [9] [8] [10] can provide a more detailed overview in M2M.

2.3 *Big Data*

Storing, processing, accessing and managing vast amounts of data with unreliable or complex structure all fall in the term Big Data. Although the term refers only to a large amount of data, the challenges that come with handling and using of it also come hand in hand with this term. Big data is often described with the 4 or 5 Vs of Big Data (depending on different sources) as can be seen in the works of [11] [12] and can be seen in Figure 2.

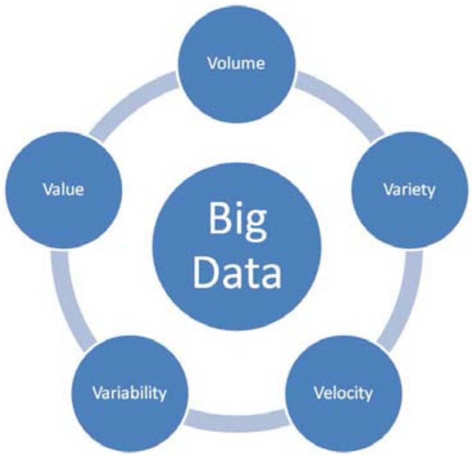


Figure 2. The 5 Vs of Big Data [13]

Volume refers to the amount of information or data that is being generated and needs to be handled. Because of the sheer amount or just the type, traditional systems (referring to RDBMS) don't offer appropriate solutions and different methods/methods need to be taken into account. The authors of [11] predict that the size of the data being generated to reach the range from petabytes to petabytes.

Variety refers to the data's structure. It can vary from a traditional sense of well structured and predictable data to semi-structured or unstructured and/or changing, coming from a variety of resources like Documents, email, Web Pages, Sensor Networks, Social Media etc.

Velocity refers to the rate of data flow. This is quite an open definition encompassing both the rate of data coming from different sources, but also rate of data flow in general.

Variability refers to the inconsistence of velocity in general. These is an issue that is hard to deal with and is best understood when thinking of the usage of social networking.

Value User can run certain queries against the data stored and then user got important results from the filtered data obtained and can also rank it according it the dimensions.

The authors of [11] also a added **Complexity** to this as an important and sometimes forgotten aspect referring to the complexity of linking, matching, cleansing and transforming of data which is coming from various sources.

Along with fundamental challenges in Big Data, regarding functional issues, security has to be taken into account if implementations are to be realized. The biggest security challenge in Big Data is as identified in [14] is the protection of user's privacy. Vast amounts of personal identifiable information (PII) that are stored in unstructured databases (NoSQL) means that the location of all information is not always known unambiguously and even direct access is somewhat abstracted. Leaving everything as is and without applying security measures such as Access Control leaves things open for abuse.

2.4 *NoSQL*

NoSQL, or otherwise known as "Non only SQL" refers to databases or data management systems which are designed to handle data management problems where conventional RDBMS solutions cannot cope for various reasons [15] [11]. These problems are usually related to: handling large amounts of data and the processing of it, high number of operations, specific types of operations or needs and others. Commonly, NoSQL systems are designed for large scale data storage and parallel processing over a large number of servers. Some systems provide APIs that support SQL or SQL-like languages and convert them into native non-SQL languages and use mechanisms different to ones in RDBMS. Because they provide the ability to handle large amounts of data it usually comes at the price of fully ACID (Atomicity, Consistency, Isolation, Durability) characteristics. This can be explained by the CAP (strong Consistency, high Availability, Partition tolerance) theorem which can be seen in more detail in [16], and because of this most systems can be described as BASE (Basically Available, Soft-state, Eventually consistent).

2.4.1 Types of NoSQL databases

NoSQL databases can be classified in four basic categories as done in [16].

- Key-Value stores
- Document databases (or stores)
- Wide-Column (or Column-Family) stores
- Graph databases

Key-Value stores

These storage systems are organized in simple, standalone tables organized like "hash tables". The items stored in tables are key-value pairs where the key is a alpha-numeric identifier and the value is one or a set of values associated with that identifier. As the organisation of table is a "hash table" the limitation that is present is that they are usually limited to only "exact match" type of queries or allowing "<,>" type of operations with a significant reduction in speed. On the other hand read operations are very fast, as to be expected from a hash table data set, and because the keys can also be viewed as the addresses of the value wanted to be retrieved, even data from the same table can be distributed over several locations so these storage systems linear characteristics regarding scalability.

Document databases

Document based storage systems, as their name implies, are designed to store data in documents. They use standard data exchange formats such as XML, JSON or BSON to store the data in documents and as can distribute these documents on multiple locations. These are considered scheme-less databases because the storage format or storage data structure can be loosely defined. Single columns or single data entries can house hundreds of values and the number or type of values stored can vary from row to row. These are good for storing and managing big collections of documents containing significant amounts of data like text documents, emails, XML documents or objects containing large amounts of values and data. Along with that they are also convenient for storing sparse data collections because of their schema-less data structure. This means that the usual filling out with null values (that is traditionally done in RDBMS) is not necessary and means that the overall amount of space used is correlated to the amount of data stored inside the database. These solutions offer great scalability, unlike key-value based stores allow multiple "< >" types of comparators and both keys and values are fully searchable. Although they can offer MapReduce [17] features they tend to have slow response times to queries. This reason is because fetching data with multiple set parameters for values means reading and parsing data from whole documents.

Wide-Column Stores

Wide-Column (or Column-Family) stores are somewhat in between document based and key-value based storage systems. They have a structure similar to a key-value structure but allow multiple values and require at least one identifier column which fills the role of a primary key. They can form multiple indexes upon other values and allow "equal type" comparisons over the value attributes. Because of the similarity to key-value based systems they share the same faults regarding "< >" types of operations. The similarity to document based systems comes because they are distributed. The key value can be used to distribute data from a table to multiple locations. This offers good characteristics regarding scalability. Reading and writing operations are fast so they are specially suited for MapReduce operations and parallel processing of large amounts of data which is their main purpose and use.

Graph Databases

Graph databases, by basic concept are relational databases but still are very different from RDBMS in the sense that they also have relations. The relations themselves can be considered as more important because these are used when we mainly need to store date regarding the relationships and dependencies between objects rather than information about the objects themselves. They store data similar to object-oriented databases as use objects as network nodes which have relationships (edges) and properties or object attributes stored as key-value pairs. The relationships can also have different attributes or properties attached to them. They are suited for storing and visualizing data regarding graphs, networks etc.

2.4.2 General overview of the technologies available at the moment

This section contains an overview of the current "popular choices" in NoSQL data storage and management systems, a brief comparison given from results provided by outside sources and general conclusion.

Hadoop

This Apache project is a platform for developing open-source software for solving scalability and processing problems in the context of large quantities of data. Data storage supported by Hadoop falls in the wide-column paradigm family of NoSQL systems. The platform has a large developer community and many open projects and because of that it is one of the biggest and fastest evolving platforms. On the Hadoop website it is stated that "The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing" [18]. It is also stated that it is essentially a framework that allows distributed processing across clusters of computers, it is designed for scalability and can easily be scaled from one server to thousands of machines, it handles failures on the application layer and as a result provides a highly-available service on top of clusters, regardless of individual failures on machines inside the cluster.

These are the projects main modules as stated on it's official webpage [18]:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

The Hadoop ecosystem of open source is built in a way that allows easy porting and migrating between different storage solutions and also data processing solutions. Therefore a project that is being developed over Hadoop has a bigger value and feature set than the value/feature set than of that single solution. Many additional options are available because of the ecosystem that it is a part of which therefore adds to the value and possible number of implementations. Also, as most Apache projects, it has a large development community so updates and improvements are frequent.

Apache HBase

Apache HBase is one of the projects built on top of Hadoop, more specifically the HDFS. It falls in the wide-column family of NoSQL database systems and is a distributed database system. As stated on the official pages [19] it is more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc. The way it is built allows it to have linear scalability characteristics and because it is a part of the Hadoop ecosystem of open source projects it is also very modular as the data stored can be used by other data processing systems and it can also be migrated to other solution if there is a need for it.

Most notable HBase features as stated on the official website [19] are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

As HBase is built on top of HDFS it provides additional functionalities. As HDFS is a general purpose file distribution systems it has it's limitations regarding usability ("from a developers perspective"). HBase stores data inside a tables and rows. This is familiar to anyone who worked with standard RDBMSs although this is almost the only similarity.

Apache Hive

Hive is also one of the open source projects developed on top of Hadoop. It is a mainly a distributed data warehouse system. The list of organisations using Hive as stated on the official website [20] include: eBay, Facebook, LinkedIn, Spotify, Taobao, Tencent, and Yahoo!. As an open source project, Hive has a strong technical development community working with widely located and diverse users and organizations. Hive was originally designed as a translation layer on top of Hadoop MapReduce. As a query language it has it's own variant called HiveQL (Hive query language) that will be familiar to anyone already familiar with SQL. It also allows for easy use of the MapReduce framework for more complex analysis and it can be extended with custom scalar, aggregation and table functions (UDFs, UDAFs and UDTFs). It does not offer real time queries or row-level updated and as such is best suited for batch style jobs (over large sets of data). As stated on the official website the main advantages of Hive are [20]:

- scalability (scale out with more machines added dynamically to the Hadoop cluster),
- extensibility (with MapReduce framework and UDF/UDAF/UDTF),
- fault-tolerance
- loose-coupling with its input formats.

Apache Spark

Spark is also an Apache open source project since 2010. It is a fast processing engine that has some advances over the standard Hadoop MapReduce jobs. It utilizes in memory processing and data caching for faster performance compared to MapReduce. The official site states [21] that it is 100x faster in memory and 10x faster on disk than MapReduce. It is designed to access data from other Apache Hadoop projects like HDFS, Cassandra and HBase and to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning. From the period of January 5th 2014. to January 5th 2015. it had 6906 commits and 419 contributors [22] making it the most (or one of the most) active among Apache and Big Data open source projects in general. It supports a variety of popular development languages including Java, Python and Scala. This project therefore is already a great solution for many large data processing applications and has great promise for future developments.

Apache Cassandra

Apache Cassandra Is again a Apache open source project in the Hadoop "ecosystem" of projects. It is mainly a data warehouse and it falls in the row-oriented wide-column family of NoSQL database systems. Same as other projects it has support from other systems in the Hadoop "ecosystem" as the data from Cassandra can be used by, for instance Spark, and processed. Porting/ migrating from other sources should also be easy. It is suited for storing large amounts of data as it also has linear characteristics regarding scalability. Main benefit of using Cassandra is a SQL like query language called CQL ("Cassandra Query Language") and functionalities like column indexes allowing more efficient storage and data manipulations depending on the structure defined by the user. Cassandra is very suitable for environments such as storing sensor data as it is scalable and has proven fault-tolerance on commodity hardware or cloud infrastructure [23]. The distribution of data across nodes is primarily organized according to the value of the primary key. Data which has the same value of the primary key will be located on the same node allowing fast reads. This also means that related data can easily be controlled and stored in one node, therefore allowing for fast access for data processing if needed.

MongoDB

MongoDB is one of the most popular choices in the document based NoSQL family of database systems. It stores data in a JSON like format called BSON in documents and because of this is very well suited for object mapping and storing unstructured data. Storing data in mongo consists of defining objects with attributes that contain data and unlike other NoSQL systems like most wide-column solutions it doesn't suffer the problem of knowing the structure beforehand. The structure can be changed at any point and as long as the implementation (the application using MongoDB) can cope with this, problems shouldn't occur.

Another good point of MongoDB is that it is well suited for use on a large variety of machines, not necessary on high performance ones. A disadvantage on using MongoDB is that it is somewhat slow regarding reading and modifying the data compared to for instance wide-column solutions but at its main purpose is to be used in implementations where the data structure/schema is frequently changing and/or unknown at the beginning.

Some of the most significant features/characteristics of MongoDB [17] [24]:

- Indexing - primary and secondary indexes are available. The benefits of indexing are the same as those found in traditional RDBMSs.
- Advanced queries - MongoDB supports range search, field search, regular expressions and other functionalities commonly lacking in other scalable NoSQL systems.
- Replication - making replicas increases the availability of data. replicas can either have a primary or a secondary role similar to a master-slave methodology.
- Load balancing and fault tolerant - using sharding based on user defined shard keys, data collections are split into ranges and distributed. The user chooses a shard key, which determines how the data in a collection will be distributed. This can be run over multiple machines and the shards are also replicated and therefore providing fault tolerance.
- File storage - as MongoDB is a document-based NoSQL distributed database system it can also be used as a distributed file storage system. This functionality is called GridFS and it is included in MongoDB.
- Aggregation - MapReduce jobs can be used for aggregation purposes. For instance achieving the usual GROUP BY functionality from SQL in RDBMSs.

Neo4j

Noe4j is currently one of most popular from the graph-based NoSQL family of database systems. It mostly useful in applications where relations between entities are the focus of implementation. It is used by a lot of companies and is supported on many platforms and frameworks so easy integration of Neo4j in almost any application in need of a graph database should be easy. This is maybe the biggest benefit of Neo4j over other possible products along with the large set of functionalities and ecosystem of tools and libraries.

Neo4j most significant features [25] [26]:

- It has a SQL like query language called CQL ("Cypher Query Language")
- It follows Property Graph Data Model
- Advanced features like: Indexing, UNIQUE constraints, transactions, ACID compliant
- It uses Native graph on disk storage with Native GPE(Graph Processing Engine)
- It supports exporting of query data to JSON and XLS format
- Support for many platforms, languages, simple to use APIs...
- High-speed traversals in the node space

Disadvantages Neo4j are mostly regarding selling strategies, expenses, the lack of features in the free "Community" version and issues with dual licenses. Regarding technical disadvantages there aren't many. Compared to other types of databases like wide-column it is les scalable and not suited for storing large amounts of data which is to be expected considering all of the features and that it's a graph-based NoSQL database system.

OrientDB

OrientDB is a newer NoSQL system (since 2012). This data management system is a mixture of document-based and graph-based NoSQL families. It stores data similar to document-based systems like MongoDB and combines it with the relations found in graph databases like Neo4j. On the official site of OrientDB [27] they have direct comparisons to MongoDB and Neo4j and state clear advantages in comparison to both alternatives. it combines advantages from both graph and document based solutions solving most of the problems by adding relations and fast traversal over relations to classic document-based solution and flexibility, scalability and sharding to a graph-based solutions. Therefore this solution has much promises and can be applied to a wide variety of implementation problems and seems like a perfect blend between MongoDB and Neo4j and a good alternative to both of those and traditional RDBMSs.

Most significant features include [27]:

- Combines best features from document and graph based systems
- OrientDB SQL - supports SQL with some extensions for handling trees and graphs
- Multi-Master Replication - sharding, fault tolerance, availability, scalability...
- ACID Compliance - fully ACID transactions across distributed data storage system
- Community Edition is free - free even for commercial use and contains all of the most significant features as the Enterprise edition accept customer support, backups, profiling and similar features

Redis

Redis is a key-value based NoSQL data store. Unlike other solutions it stores data directly in memory and uses the disk's storage only for persistence, so reading and writing is extremely fast but storage space is limited to memory. It has master-slave replication and can support any number o slaves [28]. One of the more significant advantages over other systems is the supported rich set of data types including: strings, lists, sets, hash tables, sorted sets and others.

Other features that are stated on the official website include [29]:

- Transactions
- Pub/Sub
- Lua scripting
- Keys with a limited time-to-live
- LRU eviction of keys
- Automatic failover

Redis is a simple to integrate solution not only for specific implementation situations and use-cases but can be viewed as a way to speed up fetching and modifying small frequently used parts of data. Problems regarding the storage and usage of data can usually be separated into 2 categories. In one category there would be a large quantity of data that is rarely queried or data that needs to be processed and the other, small quantity of data that needs to be queried almost always before other queries are to be executed like for instance authentication information.

2.4.3 Security issues in NoSQL

NoSQL databases solve many functional problems that conventional RDBMSs have providing greater scalability, flexibility allowing applications to better scale-out and/or utilize different kinds of data. The distribution of data is commonly done by utilizing sharding mechanisms and the usual RDBMS ACID properties are exchanged for BASE properties. While this provides good and usable features and characteristics, it bring to the surface a lot of security issues in need of dealing with. The authors of [30] [31] [32] analyzed a few of the most popular NoSQL choices and made security analysis' of them. They evaluated security features that they use and mention features that are still lacking. Areas that were looked at can be divided into: authentication, access control, data encryption, secure configurations and auditing. The database systems that were analyzed include: MongoDB, CouchDB, Redis, Hadoop, HBase, Cassandra, Redis and Couchbase Server. Although every one implements some features, all of them lack a complete set of security features. MongoDB seems to have the strongest security feature list and Redis has almost no security features. It is easy to see that the main focus of these solutions is performance, leaving security issues to be solved on the application and maintained and configured by system administrators.

2.4.4 Performance studies and comparisons

Comparisons and performance evaluations given by the authors of [16] [33] [12] [34] [35] give a good picture of the current state and capabilities of the most used NoSQL database systems. It has to be noted that the NoSQL databases taken into account were mainly either document or wide column based ad these are the best suited for storing and handling large amounts of data. The performance test included measuring latency in scenarios with different ratios of read/write/update operations. Compared to RDBMSs the results indicate that NoSQL systems have greater capacity and can cope with more operations which is to be expected. After a certain number of operations latency in RDBMS increased significantly (exponentially) while in NoSQL systems suffered little. Results indicate similar performance inside the different NoSQL paradigm families. MongoDB (document-based) was shown to be somewhat slower than Cassandra or HBase (wide-column based) with bigger workloads which is to be expected because of the differences between their storage systems and functionalities they offer.

2.4.5 NoSQL Conclusion

NoSQL systems provide solutions in the areas of IoT and Big Data. Same as in other areas, the interest of large Internet companies such as Google, Amazon and Facebook pushed developments, and because of this there a lot of good solutions available. NoSQL solves problems that conventional RDBMS cannot cope with or are not flexible enough or adapted for. Evaluations regarding data storage, handling and additional features of these solutions suggest that security is still an issue and may be the next significant improvement. Along with this new combined solutions similar to the document-graph mixed solution OrientDB utilizes could be the next step in NoSQL. Performance evaluations (regarding reads, writes, updates...) indicate that improvements are always needed and welcomed but are at a level that is satisfying for most needs. This means that the choice of the database system that is going to be used in an application shouldn't be done from a performance perspective, but from a more general and data type oriented perspective along with evaluating the ecosystem that the main solution is a part of. More significant improvements could come in the area of data processing and analyzing. The Apache Hadoop ecosystem of open source projects is currently the most fertile ground for improvements in this area.

2.5 *Access Control*

Access Control is a general term that can be described as a way of securely granting, limiting or denying access to resources therefore protecting the resources from potentially malicious parties. In this section, a few of the most significant methodologies of enforcing access control will be describe, a general overview, comparison and evaluation will be given and some available solutions will be explored and evaluated.

2.5.1 Types of access control

In this section a few of the most important and significant types of access control will be described and evaluated.

DAC and MAC

DAC (Discretionary Access Control) is a type of access control that enforces a evaluation criteria (or policy) that mainly restricts access to the resource owner or group and all control over that resource including passing access to other subjects, and is commonly done automatically or indirectly. The users/resource owners have control over the access to the resource. An example would be in operating systems. Users that create files have ownership over them and can limit access to themselves or allow access to other users. DAC is often compared to the MAC (Mandatory Access Control).

MAC is a type of access control where a external entity like an administrator decides on giving or denying access to resources. A example would be database management systems. They allow access to resources depending on the permissions users have. The system administrator has the ownership over all resources.

IBAC

IBAC (Identity Based Access Control) is based on giving access to a resource depending on a users identity. Implementations often become RBAC because identities are grouped and the groups can also be grouped as roles. An example of true IBAC are simple username and password based authentication systems such as signing up and logging into accounts on websites. IBAC is simple to implement but has limited applications because of limited flexibility regarding modifying access to resources for a number of users. Most adequate implementations would be as mentioned before, password (or authentication card, fingerprint...) based authentication systems that limit access to a resource which could be a account, secure room, etc.

RBAC

RBAC (Role Based Access Control) is an access control system where users are assigned to groups or roles which have access to resources. An administration entity has power over changing access policies for resources. This solution is one of the most popular because of the simplicity of implementation, intuitive way of working, good flexibility and intuitive administration. Different roles have different authority levels or different areas of authority. Depending on the implementation, the roles can be organized in a hierarchy tree or a simple independent list of roles or groups. Almost every website system has a RBAC system implemented limiting access to information, changing and adding data, etc. Users are typically divided into groups with different levels of access depending on the type of user (buyer, seller, visitor...) and administrators are divided into different levels and areas for efficient maintaining of the system. A problem that occurs with this kind of system is that a large number of roles create a hard system to administer and maintain. An administrator needs to have deep understanding of each role and dependencies on other roles to successfully maintain the system.

ABAC

ABAC (Attribute Based Access Control) is a type of access control that evaluates attributes to decide if a user has access to a resource. Attributes are typically divided into three categories:

- subject - user attributes (examples: age, postal code, IP address...)
- object - resource attributes (examples: type, value, age...)
- environment (examples: day of the week, hour of the day...)

Policies contain condition and after comparing these attributes against policies the access to a resource is granted or denied. RBAC can be viewed as a subset of ABAC simply by evaluating everything based on one attribute that contains a users role. ABAC is therefore more complex than RBAC and IBAC and is typically more difficult to implement. Another issue is that the policies that contain the conditions is not as intuitive and RBAC and therefore more complex for an administrator to understand. The maintaining of large RBAC systems usually require a large amount of "hands on" modifications and monitoring or a very complex and well built system to change roles automatically. If some attributes of a user change or environmental conditions change, a pure RBAC system is not flexible to change automatically. Although ABAC can be more complex to build the benefit is that is much more flexible than RBAC. ABAC policies can specifically define which actions are allowed on which resources depending on the users, resources and environmental attributes. For instance, if a user becomes older than the legal age limit they can automatically have additional access granted without changing the users role. Subjects can also have a role attribute and mainly function as a RBAC system and have policies with attributes as additional access control options that can be put in place just by adding a policy. ABAC has its place in systems that require a long term solution, complex policies and low maintenance over the policies.

2.5.2 OASIS, XACML and JSON

XML (Extensible Markup Language) is a markup language that is used to format data in such a way that it can be readable by both humans and machines. It is defined by a set of rules for encoding documents by the W3C's XML 1.0 Specification. Although the focus of XML's design is on documents, it can be and is used for storing various kinds of structured data [36].

JSON (JavaScript Object Notation) is a lightweight data-interchange format. Same as XML it is readable by both humans and machines. Although it was built around JavaScript it is completely language independent it is ideal for mapping information stored in objects (on object oriented languages) or structures and therefore is best suited for exchanging data between languages [37].

OASIS is a non-profit consortium produces open standards in the areas of security, IoT, cloud computing, energy, content technologies, emergency management, and other areas. It has more than 5,000 participants representing over 600 organizations and individual members in more than 65 countries. The goal of OASIS is to provide standards which offer efficient and open solutions for common problems the areas mentioned before, drive development and building of standardized open source solutions that can easily be used by many, therefore accelerating innovation and development in general [38].

XACML (eXtensible Access Control Markup Language) is a declarative access control policy language implemented in XML created by OASIS. It defines a way to evaluate requests for resources according to rules defined in policies. Put simply it is a taught out and standardized solution for implementing access control in software applications. It provides common ground regarding terminology and workflow between multiple vendors building implementations of access control using XACML and interoperability between the implementations. It is primarily based on ABAC but can also implement RBAC as RBAC can logically be viewed as a subset of ABAC as explained earlier in this document [39] [40]. Along with the XML version there is also a profile utilizing JSON. That profile utilizes the latest 3.0 version of XACML and as stated in the official document of the JSON profile [41] is equivalent to the standard XML. Requests and responses can be translated in either direction (and back) without loss of information and equivalent requests result in equivalent responses.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
PolicyId="smartiedissertation:accesscontrol:policy1" Version="1"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
        <Description>Policy for driving</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">drive</AttributeValue>
                        <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="true"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Rule RuleId="smartie_dissertation:accesscontrol:policy1:rule1" Effect="Permit">
            <Description>PERMIT - People over 18 to drive.</Description>
            <Target/>
            <Condition>
                <VariableReference VariableId="13245612653148"/>
            </Condition>
        </Rule>
        <VariableDefinition VariableId="13245612653148">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-
equal">
                <Description>The person accessing is over 18 years old</Description>
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
                    <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject" AttributeId="urn:oasis:names:tc:xacml:1.0:subject:age"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="false"/>
                </Apply>
<!--           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">18</AttributeValue>
-->
                   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
                      <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment" AttributeId="urn:oasis:names:tc:xacml:1.0:environment:age-limit"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="false"/>
                      </Apply>
               </Apply>
        </VariableDefinition>
        <Rule RuleId="smartie:dissertation:accesscontrol:policy:rule:default" Effect="Deny">
               <Description>DENY - default.</Description>
               <Target/>
        </Rule>
</Policy>
```

Figure 3. Example of a XACML policy

In Figure 3. an example of a XACML policy can be viewed. Simply explained this policy manages access to the resource that is an action in this case and the action is "`drive`" therefore the target (defined in inside the `Target`) are all requests that ask for access for the "`drive`" action. The policy contains a rule (stated in the `Rule`, `Condition` and finally in the `VariableDefinition` with the `VariableId="13245612653148"` part) that states that if a subject's age *`subject:age`* needs to be greater or equal than the environmental variable *`environment:age-limit`*. The default result of the evaluation of this policy is *`Deny`* and is allowed only if all of the conditions are met. The `MustBePresent="false"` values in the *`environment:age-limit`* and *`subject:age`* means that that if the original request doesn't contain these values so the PDP can contact a PIP and fetch these values.

```
{
   "Request" : {
             "AccessSubject" : {
                    "Attribute" : [
               {
                 "Value" : 25,
                 "DataType" : "integer",
                 "AttributeId" : "urn:oasis:names:tc:xacml:1.0:subject:age"
               },
               {
                 "Value" : "SubjectName",
                 "AttributeId" : "urn:oasis:names:tc:xacml:1.0:subject:name"
               }
             ]
        },
        "Action" : {
             "Attribute" :
                  {
                     "Value" : "drive",
                     "AttributeId" : "urn:oasis:names:tc:xacml:1.0:action:action-id"
               }
           },
        "Resource" : {
             "Attribute" : [
               {
                     "Value" : "Porto",
                     "AttributeId" : "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               }
              ]
           }
   }
}
```

Figure 4. Example of a XACML request in the JSON format

In Figure 4 an example of a request can be seen. As it can be seen, the JSON format of XACML is somewhat easier to read and is less cluttered than the original XML variant. This request simply states that it wants to execute the action `drive` on a resource with the id `Porto`. If the age limit is lower or equal than 25, this request will pass and will be allowed.

2.6 *SMARTIE*

SMARTIE (Smart City) is a European project with the goal of solving security, privacy and trust issues in IoT, in a Smart City implementation. Partners include companies, universities and cities from Germany, Serbia, Spain, Portugal and UK. As stated on the official website [42] the project officially started on September 1st 2013. and is scheduled to end on August 31st 2016. and has a total budged of 4,862,363 € with the contribution from EU in the amount of 3,286,144 €.

2.6.1 Brief Overview

Vision [42]

The vision of SMARTIE is to create a distributed framework to share large volumes of heterogeneous information for the use in smart-city applications, enabling end-to-end security and trust in information delivery for decision-making purposes following data owner's privacy requirements. A secure, trusted, but easy to use IoT system for a Smart City will benefit the various stakeholders of a smart city: The City Administration will have it easier to get information from their citizens while protecting their privacy. Furthermore, the services offered will be more reliable if quality and trust of the underlying information is ensured. Privacy and Trust are a key prerequisite for citizens to participate in Smart City activities. A Smart City can improve the Smart and Comfort Live of their citizens enormously. Enterprises benefit from the securely provided information. They can optimize their business processes and deal with peak demands introduced by the dynamics of the Smart City. Furthermore, they can offer more tailored solutions for their customers based on the status of the Smart City.

Main goals [42]

- Understanding requirements for data and application security and creating a policy-enabled framework supporting data sharing across applications.
- Developing new technologies that establish trust and security in the perception layer and network layer.
- Develop new technologies for trusted information creation and secure storage for the information service layer.
- Develop new technologies for information retrieval and processing guided by access control policies in the application layer.
- Demonstrate the project results in real use cases

Key Challenges [42]

The idea of the IoT brings new challenges regarding security and in consequence also for privacy, trust and reliability. The major issues are:

- Many devices are no longer protected by well-known mechanisms such as firewalls and can be attacked via the wireless channel directly. In addition devices can be stolen and analysed by attackers to reveal their key material.
- Combining data from different sources is the other major issue since there is no trust relationship between data providers and data consumers at least not from the very beginning.
- Secure exchange of data is required between IoT devices and consumers of their information

Expected Impact [42]

- The SMARTIE IoT platform will allow the virtualization of the functionalities of discovery, secure information access, processing and privacy-aware distribution between the consumer and producer of the data generated by the smart objects.
- It will demonstrate the applicability in scenarios linked to the green behaviour and sustainability of smart cities like efficient transport/mobility and energy management.
- SMARTIE will facilitate new companies for developing and providing services over its IoT infrastructure/platform.
- SMARTIE allows heterogeneous and multiple source of data to interact in reliable and secure manner providing third parties developers in Europe to enter the Smart Cities area and in that way increase the share of the IoT market.

Use Cases [42]

1. **Frankfurt/Oder (GER)**
    - Traffic management with the possibility to influence real traffic.
    - Focus on authentication, trust, data security, interoperability
2. **Murcia (ES)**
    - Monitoring energy efficiency in the campus
    - Users can interact with the system to improve energy efficiency.
3. **Belgrade (RS)**
    - Provide smart transportation using location of busses and travellers
    - Focus on data security and privacy using developed access rights and policies

2.6.2 Conclusion

There are a number of smart city project currently being developed but unlike SMARTIE most are focused on specific aspects and specific test scenarios and rarely grow bigger after they are complete. SMARTIE is a solution for smart cities that unlike other efforts is being developed with security concerns and standardisation in mind. The fact that it is founded by the European Union gives even more significance to the project because if successful, it will surely be utilized in many European cities. Along with having the focus on solving this problem for cities today, the project also focuses on utilizing standards, stimulating innovation and producing improvements and developments in the area of IoT.

2.7 *Related work*

Axiomatics

3 Solution project description

This section will describe in detail the work that has been done along with a short description of the development process.

In Section 3.1 the solutions architecture and workflow will be explained along with design choices and technologies used, Section 3.2 will give a description of every major component, Section 3.3 will present the proposed implementation scenarios, Section 3.4 will give a brief overview of development process and Section 3.5 will present a critical overview and potential improvements.

3.1 *General architecture, design choices and technologies used*

After extensive research and regarding the technologies currently available, a number of design choices were made and a solution architecture is proposed.

The NoSQL databases are used are Cassandra for storing sensor data and Redis for storing policies. Cassandra was chosen because of its good balance between scalability and response time characteristics and Redis for its speed of fetching data. These choices were made after significant research was done on the performance characteristics of these databases compared to other options, which can be seen in Section 2.4. Cassandra proved to be the most appropriate solution for storing data and Redis for the storing of policies. Cassandra also offers compatibility with other Hadoop systems because it is a part of the Hadoop eco-system which could be very beneficial because of data processing options and possible migration to other systems if the need for that comes up. Redis is on the other hand a smaller and simpler solution which stores data in memory, because of which the response is so fast. The drawback is limited space for storage which is not of a great concern because the space required for storing policies is not large. The Cassandra NoSQL database used in the solution is used just for the testing purposes because the intention of the solution is using it for enforcing access control on a variety of resources and data. Cassandra is also used for storing attribute data. This is also done mainly for testing purposes regarding the PIP. Data Manager modules are placed in front of both databases to control the communication and management of the databases. These managers therefore are the only entity that control and survey all the interaction with the databases, they limit the type of actions that can be executed and therefore ensure against using some unwanted ones like deleting and/or emptying tables.

The access control framework utilizes some functionality provided by the AT&T project [2] and uses it to create the all the modules needed for the framework to work: PEP, PDP, PRP, PIP and PAP. The architecture used is based on the one the one described in the standard's specification [1] and instead of having all of the functionality specified in the standard be implemented with a subset of simpler functionalities. The limitations that will be taken into account come directly from the current state of the SMARTIE project (March 2015.) which means that all or most of the use-case scenarios will be fulfilled with the overall solution being simplified making the delivery more realistic (taking in account the time constraints/deadlines for this work). Of course, the solution allows for easy upgrading and expanding of functionality therefore making it possible to have a full implementation of the XACML standard.

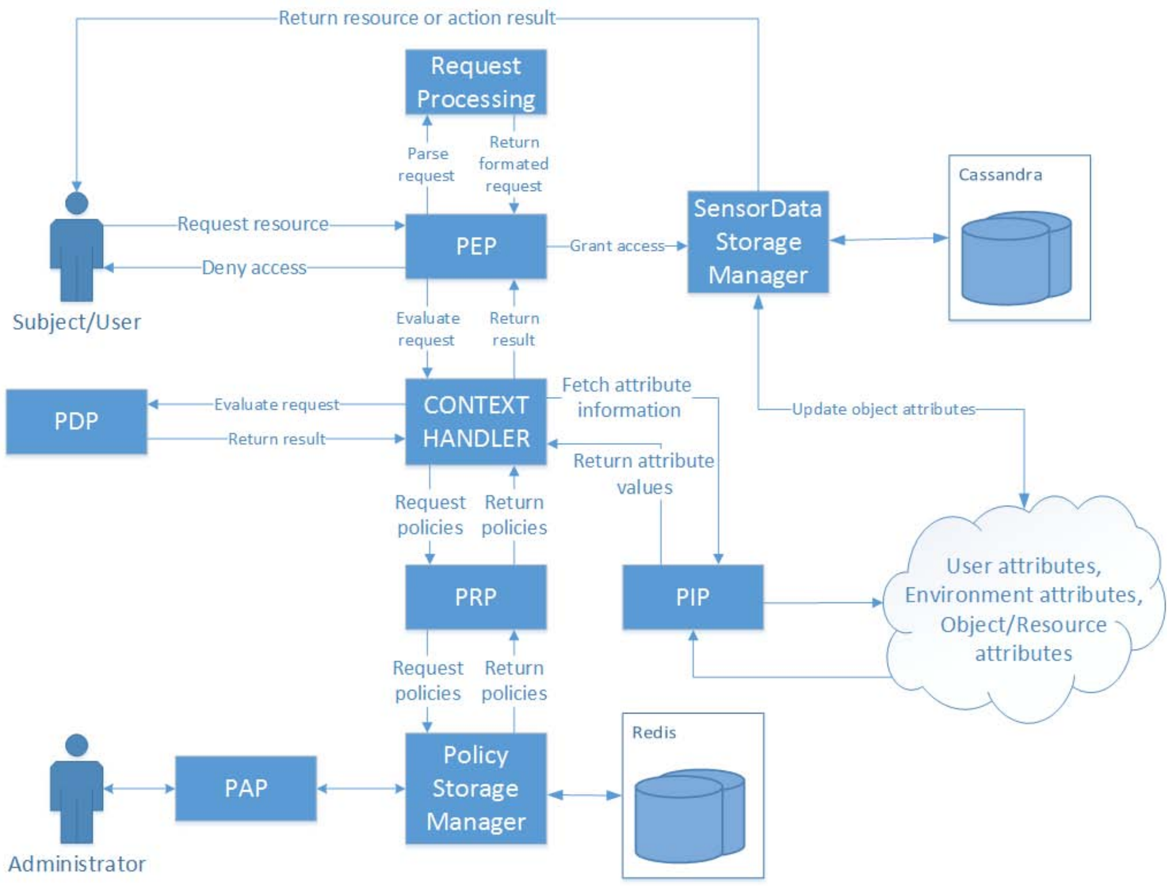


Figure 5. Architecture of the initial proposed solution

In Figure 5. a more detailed architecture of the initial proposed solution can be seen. This solution is almost the same as the proposed one in the OASIS standard [1]. The PEP is the point where the enforcement happens and the request is forwarded if a policy exists that the execution of that request. It is intended to have as simple as possible and all of the configuration, workflow managing and evaluation are done in the Context Handler and PDP. The PIP is intended to be flexible in a way that it allows connection to external services for fetching attribute information. This means that the initial request provided to the PEP doesn't need to contain all the information (attribute values) and therefore the access to those services that provide attribute data can be limited only to trusted services like this one and not to users/subjects. While developing this solutions a number of problems and the architecture was therefore modified. The main issue was the updating of the resource attributes and storing them on a separate system and the security issues regarding that connection and the synchronisation of data between the systems.

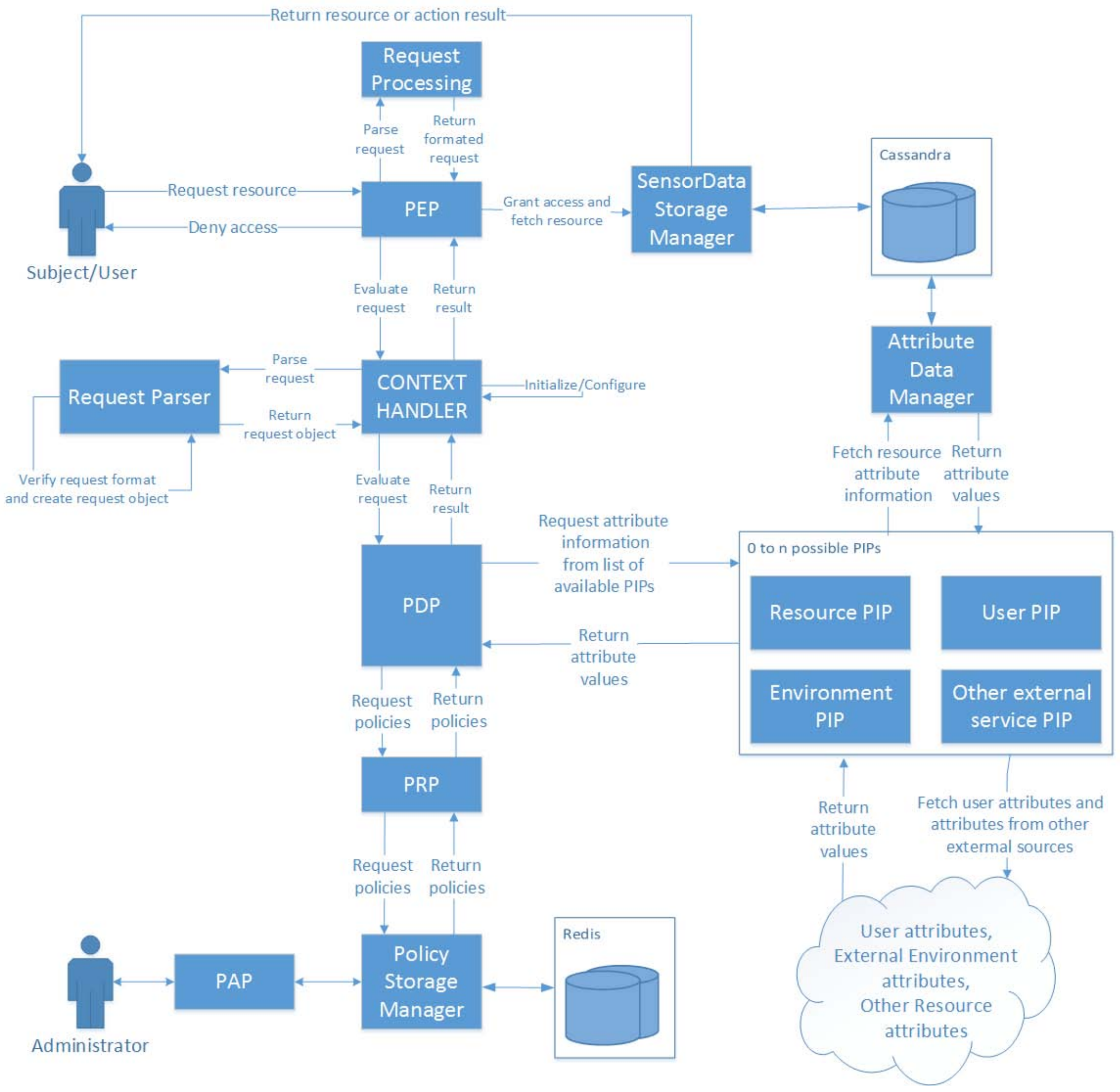


Figure 6. Architecture of the final solution

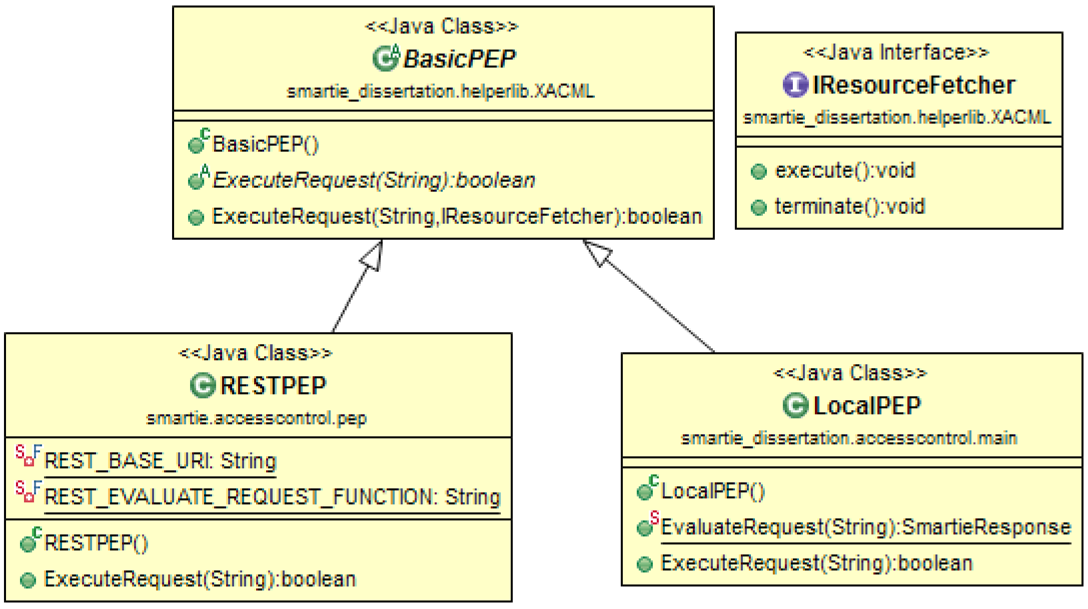
After some work was done a final architecture was settled upon with slight modification to the initially proposed, and as a result has slightly departed from to the architecture proposed in the XACML standard [1]. The final architecture can be seen in Figure 6. The changes do not change the "outside" view of the system but are more of a internal change and more refined solution. The connections to the PIP and PRP are moved from the Context Handler to the PDP so it can fetch policies and all of the attribute information as it needs, while evaluating policies. The PIP is not a single entity but rather, a list of PIPs that all have the same interface and all fulfil the same purpose of fetching attributes but because some attributes are located on different locations and need to be fetched using different services they need to implement different means of fetching that information. This allows for easy expansion of the PIP functionality and better configuration options.

3.2 *Description of components*

In this section a functional description of the developed solutions will be described along with some implementation details and database schemas that were used. The components that will be described can be seen in Figure 6. that shows the architecture of the final solution.

PEP

The PEP is the point where (as the name states) access control is enforced. This means that this point needs to be located in the system that wants to enforce access control at the exact place inside the workflow where access control is needed. It therefore needs to be built rigid enough to ensure correct execution and flexible to be implemented on various types of systems. Because of this and reasons explained in Section 3.4 the PEP can be used in multiple ways. It can be implemented by providing it with only a XACML request and depending on the response given act appropriately. This way the system that is implementing the PEP decides what the resulting action will be after the evaluation is finished. The other way is to along with the request, provide the PEP with an object that implements a defined interface `IResourceFetcher`.



<<Java Class>>
BasicPEP
smartie_dissertation.helperlib.XACML
BasicPEP()
ExecuteRequest(String):boolean
ExecuteRequest(String,IResourceFetcher):boolean
<<Java Interface>>
IResourceFetcher
smartie_dissertation.helperlib.XACML
execute():void
terminate():void
<<Java Class>>
RESTPEP
smartie.accesscontrol.pep
REST_BASE_URI: String
REST_EVALUATE_REQUEST_FUNCTION: String
RESTPEP()
ExecuteRequest(String):boolean
<<Java Class>>
LocalPEP
smartie_dissertation.accesscontrol.main
LocalPEP()
EvaluateRequest(String):SmartieResponse
ExecuteRequest(String):boolean

Figure 7. Class diagram of the PEPs

In Figure 7. the class diagram for the PEPs can be seen. The `IResourceFetcher` is used to ensure that the object provided has methods available for both the positive and negative results of the requests evaluation. With this, the PEP executes the `execute()` in case the evaluation result is positive and executes `terminate()` in case of a negative result. The purpose of this is to remove the decision making part from the system that implements the PEP and have it already built in and working. In the case of specific scenarios, the other method of simply getting the evaluation result is also available. The `RESTPEP` and `LocalPEP` should never both be available for use by another system and the intent is to have only one PEP available for implementation/integration but they don't collide in functionality. This is explained more in Section 3.4.

Databases

The databases that were chosen were Cassandra and Redis. Cassandra was chosen because it is well suited for storing sensor data and Redis because of its simple implementation and fast response. These choices were explained in more detail in Section 3.1.

In the Cassandra database a single table was used for storing sensor data. It has a simple schema with basic fields for storing simple sensor data. The use of this database is strictly for initial testing purposes and that is the reason behind the simple schema. Other scenarios involve using other databases or resources.

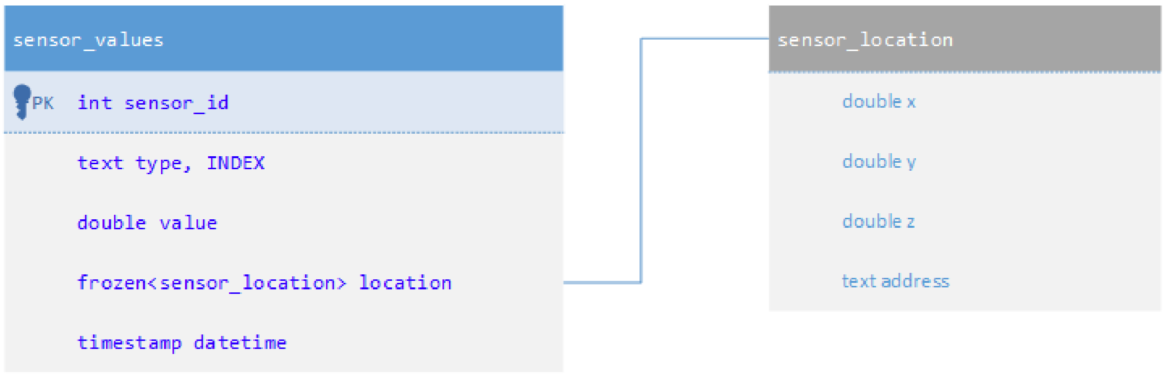


Figure 8. Cassandra database schema for storing sensor data

Figure 7. shows the schema made to store sensor data. The value of the readings is stored in the `value` column. The location is integrated as a `frozen` which means as a separate, custom structure that contains both coordinates and address values. Any of these values can be empty making it flexible. As the focus of this work isn't on analysing data the values aren't of real concern making this schema good for storing generic sensor data for testing purposes. The `INDEX` is made on the `type` column because comparison tests regarding attribute values were mainly made on this value.

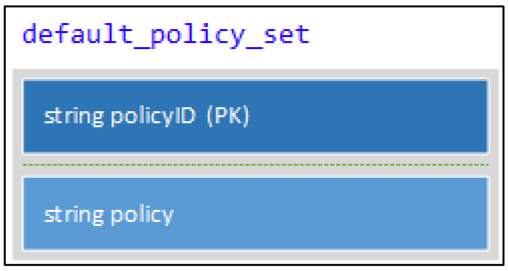


Figure 9. Redis schema for storing policies

Figure 8. shows the structure used for storing policies on the Redis database. The structure is a simple hash map with a `policyID` as a key and the actual *policy* in string format stored as the value. The `policyID` is a SH1 hash value of the actual policy. This removes the possibility of collisions happening unless the policies are exactly the same. As Redis stores data in memory the response is fast. As the number of policies shouldn't be large, the limitation of storing data in memory shouldn't be a problem. In the case of memory problems, Redis provides some options with storing both on disk and in memory or the solution could be migrated to another NoSQL database like CouchDB which has slower response but also less memory limitations.

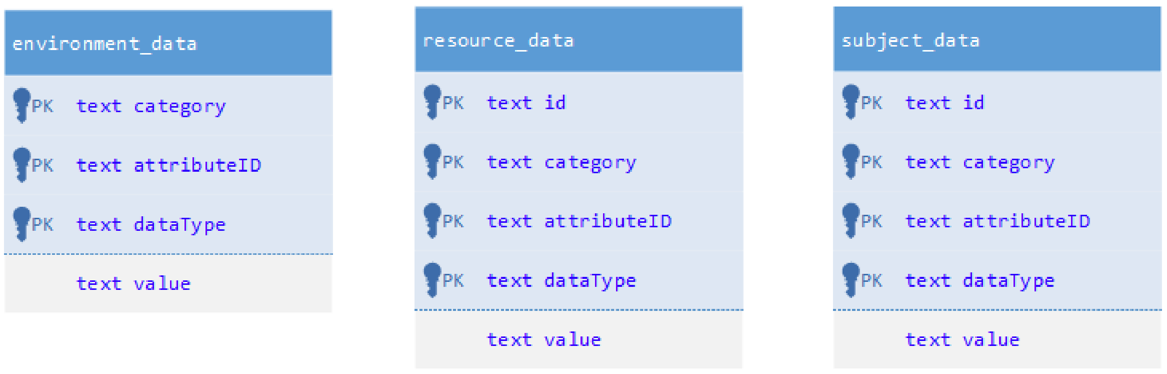


Figure 10. Cassandra database schema for storing attribute data

Cassandra is also used for storing attribute data as seen in Figure 9. The tables are constructed to mirror the fields needed when fetching attribute data while evaluating requests against policies. The fields are therefore named the same as the ones from XACML policies: `Category`, `AttributeId` and `DataType`. All columns are type string because it is the simplest, most convenient way and the of storing attribute data and the testing of the PIP's functionality. The tables for storing resource and subject attribute data have an additional *id* column because many subjects and resources could have the same type of data but would originate from different resources/subjects. The environment is considered as being a single entity and the distinction between values is done mainly done with the name. These tables could have been merged because they differ between each other in the *category* column, but because of the logical difference when considering ABAC and XACML, they were separated.

Database Managers

Database manager are entities that are placed in front of database clients. Both Cassandra and Redis have Java clients that were used for working with the databases. The database managers serve the purpose of configuring the database, schemas and limiting the possible actions to acceptable ones like adding and retrieving data. The `PolicyDBManager` and `AttributeDBManager` allow for deletion and modification of data while the `SensorDBManager` doesn't allow for those actions.

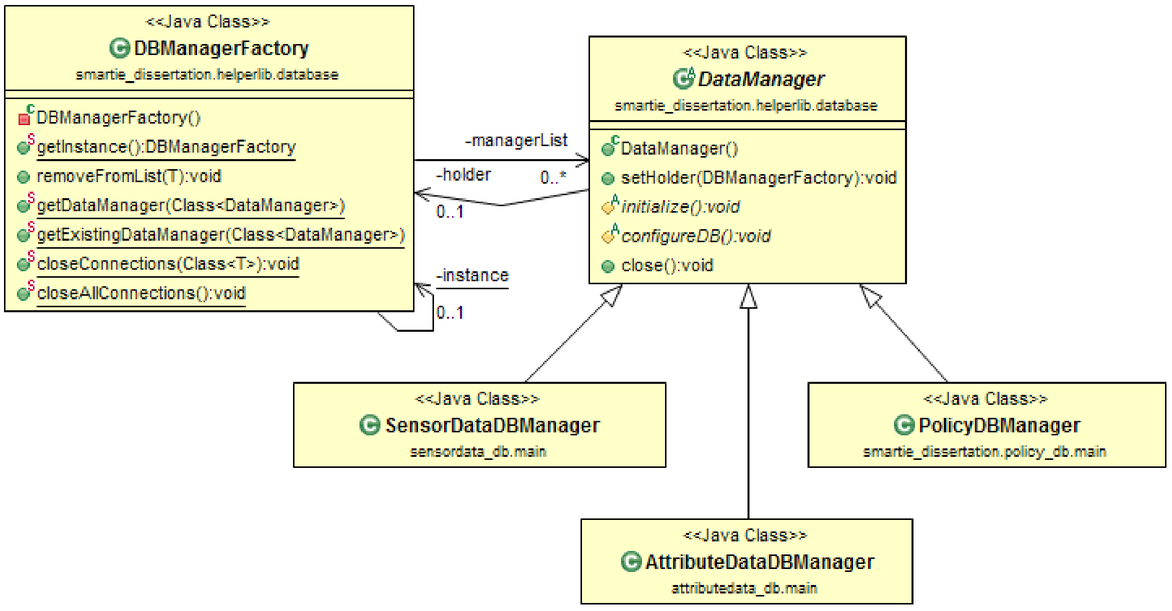


Figure 11. Class diagram for the Data Managers

The class diagram shown in Figure 10. shows that the `DBManagerFactory` holds instances of objects that extend the abstract class `DataManager`. This has been done to enable the use of `DataManagers` in a singleton fashion without having the same functionality copied to everyone. It also allows for some functionality to be added on top of the whole group and the execution to be controlled from one place (`DBManagerFactory`). This structure doesn't guarantee that there will only exist one instance of the objects a public constructor has to exist but as the intent is to get instances trough the Factory, that can be neglected.

PRP

The PRP is a simple entity that is responsible for fetching policies and delivering them to the PDP for evaluating requests. The PRP calls the `PolicyDBManager` for this purpose. It is called trough a single static function that returns the policies in a list.

PIPs

The PIP is not implemented as a single point but as a list of many PIPs. This is convenient because of the many possible ways that the PIP can fetch attributes and different places from which the PIP has to fetch attributes. This allows for modular adding and removing of PIPs depending on the particular implementation. There are 4 main PIPs that were made and used in this solution. The *LocalEnvironmentPIP* used to fetch local environmental attributes. These are basic time related attributes (current date, current time...) that were taken from and defined as stated in the OASIS XACML standard [1]. The *ResourcePIP* is used for fetching resource attribute from the database that contains sensor data. The *ExternalPIP* is used for fetching attributes (external, resource and subject) from a database that has attribute data stored. This is used to simulate fetching data from a source other than the one containing the resource and although the actual database is the same (Cassandra) the database manager is different and the data is stored in different *tables* under different *keyspace*s so it is effectively a different source. Finally the *RestPIP* is used for fetching attribute data from a REST service. This simulates fetching data from external sources. This list naturally has to be configured/modified or expanded for it to be used by other systems as these are built for testing purposes. These PIPs cover all the scenarios for generating attributes, fetching ones from a database with direct access and fetching from outside services (via REST service). These therefore provide sufficient functionality for testing but can also easily be adopted to work in other scenarios. The PIPs are an implementation of *PIP ConfigurableEngine* interface provided by AT&T project [2]. This allowed for easier utilisation of the PDP's functionality.

PDP

The PDP is the most crucial and complicated component needed for a XACML ABAC access control implementation. The purpose of the PDP is to evaluate requests using policies and returning a response which contains the decision of the evaluation. The rules defined in the standard [1] are extensive and use many operations allowing the policies to be well defined and flexible. This, of course, makes the implementation more complex. The evaluation of policies was taken from the AT&T [2] project and a implementation was built to suit the purposes of this solution. The workflow of the PDP consists of the following actions:

```
1.  Receive request
2.  Fetch policies from PRP
3.  For every policy
    3.1.  Begin evaluation
    3.2.  If attribute is missing for evaluation
          3.2.1. fetch needed attributes from list of PIPs
    3.3.  evaluate request
    3.4.  store result
4.  If a positive response exists
    4.1.  Return that response and positive result
5.  If Every result is negative
    5.1.  If there was a valid response (a policy applied for the request)
          5.1.1. Return that response and negative result
    5.2.  Else
          5.2.1. Return empty response and negative result
```

Figure 12. PDP Workflow

As seen in the workflow in Figure 12. the PDP is set to pass the request if there is at least one policy that gives it permission. The policies therefore have to be written with this in mind. All policies should deny permission by default and allow only if the conditions are met. This was made as it accommodates most scenarios and reduces the complexity of using the solution. The PDP always fetches

Request Parser

The Request Parser is an entity that checks the validity of the request sent to the PEP and converts the request from the raw string format into a Request object so it can be used by the solution. It also recognizes if the request is in JSON or XML format so the system supports the use of both.

PAP

The PAP is and entity that is used to manage the policies used. It has functionality for adding, removing and modifying policies. To access the PAP a simple web application is used as an interface on which the administrator can manage the policies used. The access to the administrator functionality is enforced by the system itself. A PEP is implemented and is called whenever a policy is trying to be removed, added, or modified. XACML policies were written and placed on the system before the PAP functionality was added and can also be managed by the PAP the same way as any other policy. This therefore brings the functionality of the system "full circle" as it can be said that it is an Access Control framework which enforces Access Control on itself.

Packages and dependencies

This section will describe the dependencies between the several parts of the developed solution.

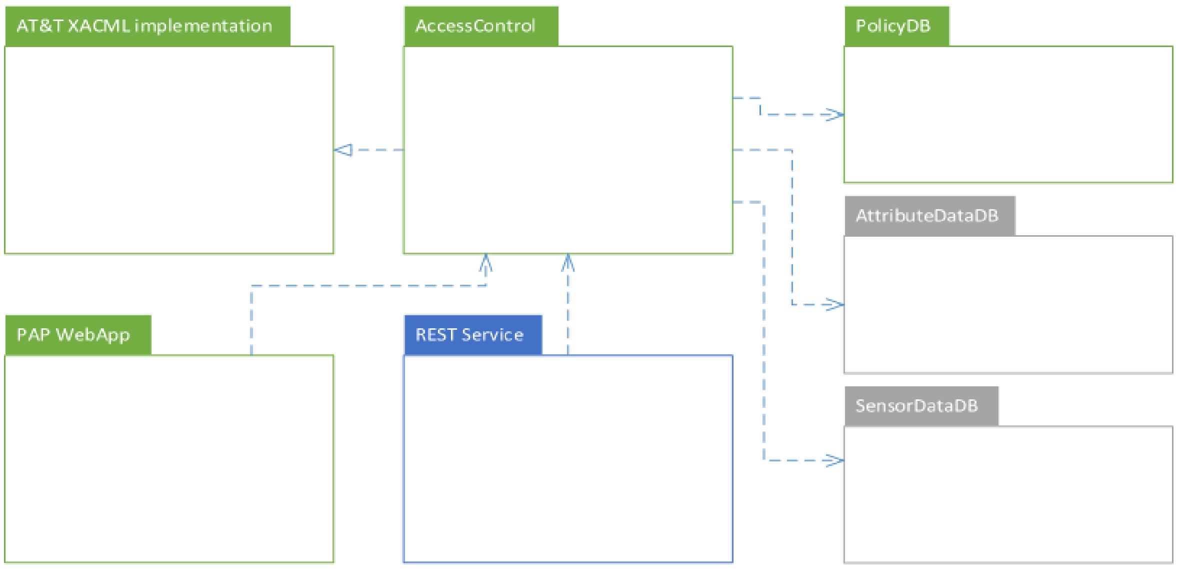


Figure 13. Project dependencies diagram

In Figure 13. the dependencies between the several parts can be seen. The different colours used in the diagram represent the different levels of "importance" as not all parts are always required. This depends on the implementation scenario and will be explained later in this section. Also, one component that is missing is a basic "Helper" component that contains useful generic classes and functionalities like a logger, basic Data Manager and basic REST Client, that are used by all parts and is why it was left out of the diagram. As can be seen the central project is the *AccessControll* project. It contains all the main functionality and components. It uses the functionality provided from the AT&T project [2]. The *policyDB* along with the Redis database it uses is an essential part of the system as it is used for storing policies. The PAP Web application although not needed for the system to function and evaluate policies provides and interface for managing policies and needs to be ran on the same system (machine) as the rest of the system. The blue colour of the REST Service signifies that it is not needed for the system to run. It's purpose is to provide an easy to get to and use interface for evaluating requests. It forwards the requests to the Access Control's Local PEP and returns the response to the caller. Because the Access Control can also be integrated and enforced locally this component isn't strictly necessary for the system to run, hence the blue colour The implementation scenarios will be explained in more detail in the next Section 3.3. The *SensorDataDB* and *AttributeDataDB* are used only for testing purpose and the connection to those has to be configured depending on the implementation scenario. The purpose of those dependencies is only for fetching attribute data. These are also not necessary if the request already contain all of the attributes needed for evaluation hence the grey colour of those components.

3.3 *Integration scenarios*

This section will describe 2 possible ways that the solution is predicted to be utilized in order to enforce access control in a target system. It will also provide an critical overview by presenting security issues and threats, and proposing solutions for solving those issues. These scenarios have some common characteristics and requirements but also differ in the benefits they offer and issues that need to be dealt with.

Integrated solution

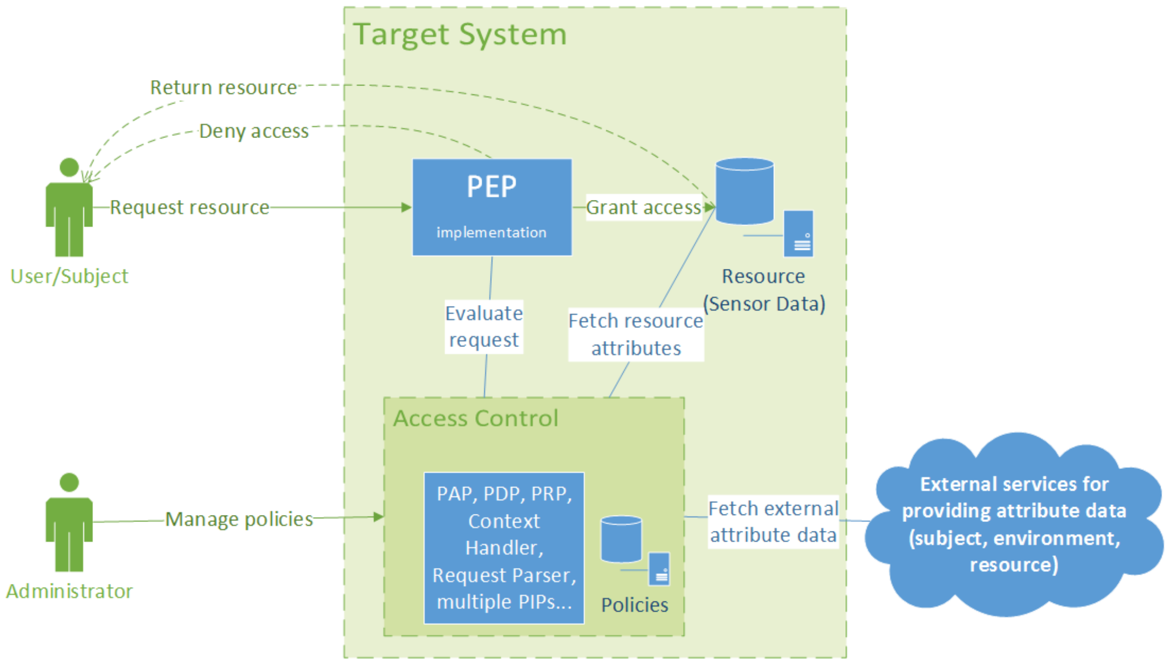

Target System
Return resource
Deny access
Request resource
PEP
implementation
Grant access
User/Subject
Resource
(Sensor Data)
Evaluate request
Fetch resource attributes
Access Control
PAP, PDP, PRP, Context Handler, Request Parser, multiple PIPs...
Policies
Manage policies
Administrator
Fetch external attribute data
External services for providing attribute data (subject, environment, resource)

Figure 14. Integrated solution scenario schema

In Figure 14. the schema of the "Integrated solution" scenario can be seen. This scenario is the first of the two integration scenarios. This scenario requires that the target system has a Redis database running and available to be used by the Access Control component. As the solution is completely integrated in the target system it allows for it to be configured very precisely for the target system. The implementation of the

PEP is done by providing it with a request and object responsible for fetching the resource. This removes the responsibility of decision making process from other entities and is a "clean way" of enforcing access control. Other that providing and configuring the Redis database for storing policies additional configurations depend on the needs of the target system as these additional components are not strictly necessary as explained in Section 3.2. If the system requires fetching of attribute data from external sources like for example subject attributes for an internal or external data source a PIP needs to be configured to connect to that system and fetch the required data. The same applies to fetching resource data from a database containing resource data. As connection to outside sources and connection directly to the resource for getting resource attributes are a security issue, these matters have to be dealt with. This matter will be analyzed in more detail later in this Section. The administration entry point is a simple Web App that offers functionality for managing the policies. Also the access is to this functionality is secured and managed by using the same PEP as the Access Control system enforces access control over itself.

Using the solution as a service

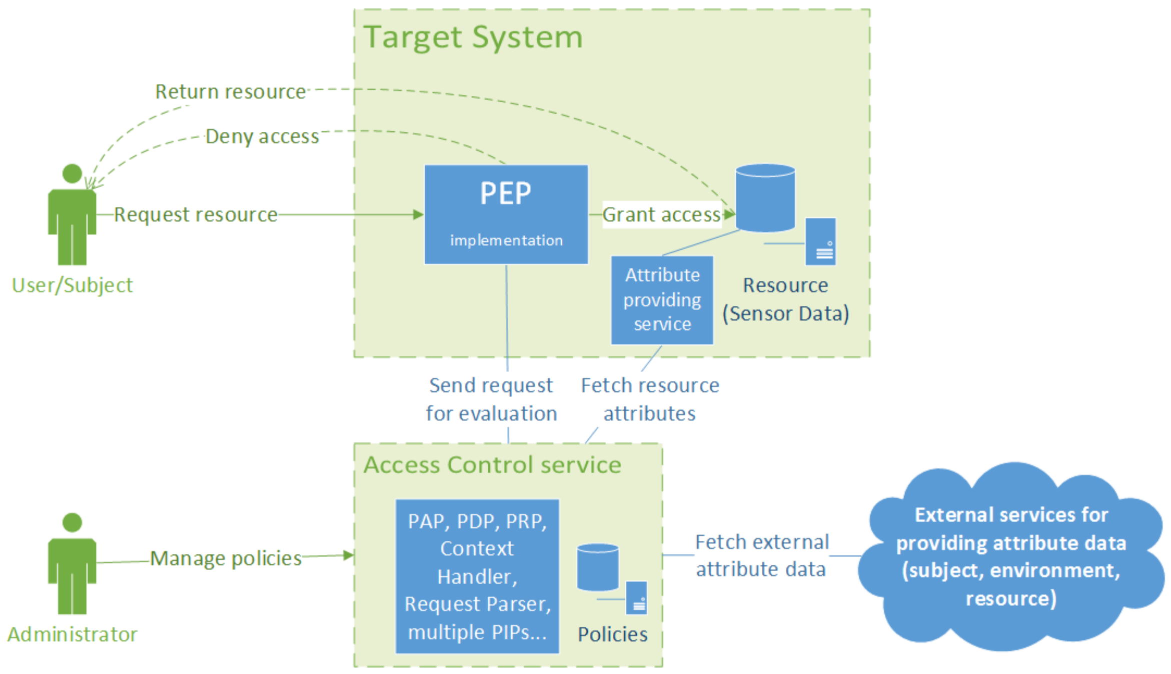

Target System
Return resource
Deny access
Request resource
User/Subject
PEP
implementation
Grant access
Attribute providing service
Resource (Sensor Data)
Send request for evaluation
Fetch resource attributes
Access Control service
PAP, PDP, PRP, Context Handler, Request Parser, multiple PIPs...
Policies
Manage policies
Administrator
Fetch external attribute data
External services for providing attribute data (subject, environment, resource)

Figure 15. Schema of using the solution as a service

Figure 15. shows the schema of the second integration scenario. The main difference between this one and the first one is that the Access Control system is not integrated into the Target System but is called from outside trough a REST service. For basic use this scenario requires minimal configuration and unlike the first one, it doesn't require a Redis database for storing policies. The PEP that is enforced and integrated has the same interface and provides the same functionality. If it needs to utilize fetching of attributes from external sources those PIPs need to be implemented and integrated in the Access Control service before and configured to connect to the correct services in a secure manner. Unlike the first scenario if the Access Control needs to fetch resource attributes the Target System needs to provide an end point (connection for a REST service) that provides a method for fetching resource attributes.

Comparison

The integration scenarios share some characteristic and differ in others. The main difference is in the fact that the first one is a locally integrated solution while the other is using the Access Control as a outside service. This difference means that the first scenario is more secure but relies on the target system, can be configured and modified to accommodate more specific uses, more precisely, but requires to have a Redis database and cannot be used in the same way by multiple systems if they are distributed on multiple machines/locations. This scenario therefore is suited for closed systems that do not require connections to outside services making it more secure. The second scenario is simpler to integrate for a simple and basic use. It relies on the outside Access Control system to deliver the decision which therefore has limitations regarding configurations/modifications but can be monitored and updated easily and it removes a lot of the responsibility from the target system. The second requires the implementing of a REST service for providing resource attribute data. The second scenario therefore comes with the more security issues regarding all the external connections that eventually need to be solved and are going to be addressed in the next part of this Section.

Security Issues and threats

These integration scenarios propose 2 methods of integration but also reveal some issues, mainly regarding security. As the purpose of this solution is to provide the means to enforce access control and therefore security, these issues must be dealt with if the solution is to be used. The main issues are regarding the way external attribute data is fetched. This is achieved through an open REST connection/service which is insecure as the client cannot be certain it is communicating to the right service, the service doesn't know if it is responding and sending data to the right client and the also there is no guarantee that the message was not tampered with. The connections that are of concern are the connection to external services providing data in both scenarios, and the connection between the PEP and Access Control Service, and the one between the Access Control Service and the Attribute providing service located on the Target System. These connections need to be secured in order for the system to be secure. A simple and effective way of securing these connections and solving these issues is by implementing the REST services over a HTTPS connection (SSL, TLS). Using this method provides the authentication to both parties involved in the communication and protects the privacy and integrity of the data being exchanged between them. This would be sufficient to solve these issues because the Server and Clients could trust they are communicating with one another and that the messages are not being tampered with. Other options like OAuth 2 and OpenID Connect are also solve these problems and additionally provide additional benefits when considering connection with other systems but this work won't go into a detailed analysis of those options nor TLS. It will be stated that every one of these solutions would be sufficient to secure the connections but OAuth 2 and OpenID Connect offer additional benefits that could be used by both the Access Control and Target system. The last issue is with fetching resource attribute data in both scenarios. Although this is not necessary a security issue it has to be mentioned as it is a potential point of access trough which sensitive data could be fetched. This point should be different than the one used normally to access resources and it should limit the access only to parts and data that are really needed for the evaluation of policies and avoid fetching large quantities of data and sensitive data. Of course, depending on the policies used sensitive data has to be accessed for evaluation purposes, and the PDP (and therefore the PEP) will not return any additional data so it isn't a true security issue. This is why the issue is not necessary a security issue but has to be addressed with caution. For the purposes of testing done in this work a different Database Manager was used to access the Resource Attributes for the database containing sensor data. Compared to the *SensorDataDBManager* it has a more limited view of the database and more functionalities that limit the actions over the database only to fetching/read actions.

3.4 *Development process*

This section will briefly describe the development process used while developing this solution. The process followed the methodology of building a basic and simplified version of the systems core early, and incrementally expand, test and refactor the existing solution until a fully functioning solution is reached. In the early stages of development the requirements for potential systems like SMATRIE were taken into account and as a result the solution's scope was determined. The resulting scope allowed for the solution to implement a subset of functionalities mentioned in the OASIS XACML standard but allow for expansion.

The solution was developed in several stages. In the initial stages the basic architecture was made and the databases were configured. The Database Managers were made to provide easier, additional control and limit the possible interactions with the databases. The databases were tested and an initial schema was setup for testing purposes. After that stage the development of the Access Control framework began. As development of a PDP "from scratch" was unrealistic in the timeframe intended for this work, a number of base projects were considered. After some consideration the AT&T git hub project [2] was selected as a base project from which some core functionality will be used. As the project was in a "beta" for it lacked documentation and made the implementation and especially the configuration very difficult. Because testing proved that some core functionality is working and because there weren't many options available at this project was chosen. The main functionality utilized was the evaluation functionality of the PDP and parsing functionality for the verifying requests and policies. After the project was tested the basic architecture of the solution was made and the development of components began. The first component that was built was the PDP. After an initial implementation was made and tested, a PRP and basic PEP were built. At this a list of basic request and policies were made. This list was used as a base which was later expanded in order to test other components like the PIPs and different evaluation operations. Basic PIPs were made and connected to the existing components. After this was complete a working prototype with all of the essential components except the PAP was made. The PAP was the last component made as it is not essential for testing purposes as the focus of the work is on having a framework which would evaluate requests and policies and not for building policies and requests. After running tests locally and confirming that the system is working correctly and giving correct responses the next stage involved the creation of a REST service and running tests from other systems. The main test that will be presented were done at this stage. The last component built was the PAP Web Application for managing policies. The last stages of development included refactoring and improving of the solution, expanding the list of PIPs, expanding on the PDPs evaluation functionality and integrating and testing with other systems. The results of the testing are shown in Section 4.

3.5 *Potential improvements/achievements*

In the current state, SmartData has access control implemented by encrypting data and allowing decryption to users that have access to it. The policies that contain the information on which users can access the data are stored together with the data itself ("sticky policies"). This means that the all data needs to be fetched in order to evaluate the policy on it. This has a drawback in case of requests that result in denying access. If a user or set of users send a large number of requests for a lot of data, for which they don't have access to, an opening for a DoS type of attack could happen. This also means that in case of changing access policies for past data means fetching all the data that we want the change to affect and updating of the policy attached to it. This solution therefore isn't flexible if policies that are meant to be updated are to be integrated.

This Access Control framework should solve this issues offering a safer and more flexible solution and could potentially be integrated as access control "before" (regarding the point in communication) the current access control point.

.... TODO: elaborate a bit more

4 Testing

4.1 *Test scenario - local scenario*

4.2 *Test scenario - real data, real IoT scenario*

5 Integration/testing with SMARTIE

5.1 *Current state*

5.2 *Proposed implementation*

5.3 *Testing 1 - Fabio, testing as a service*

5.4 *Testing2 - testing in a service (Locally)*

6 Conclusion

6.1 *Integration in other systems*

6.2 *Future work*

6.1 *Final thoughts*

Companies such as Google, Facebook, Amazon... etc. are investing a lot of time, effort and money in solving the challenges regarding IoT, Big Data, Access Control, Security in all of these areas and more because future steps in the commercial technological aspect in general point to the integration of microcontrollers or computers in general into almost every device and everyday item or can be integrated into infrastructure, factories, farms, production facilities ect. This leads to an IoT which leads to Big Data problems which are being addressed by NoSQL data management systems. Along with the fundamental problems like infrastructure and having solutions that can cope and handle everything, security and privacy of all of that data is an issue that needs to be addressed for "real world" applications to be acceptable, safe and successful. ABAC using the XACML/JSON standard from OASIS is one of many possible solution/routs that can be taken regarding the solving of this problem. Compared to other solutions it offers exceptional security if taken into account the minimum amount of human effort needed for managing and updating policies. The implementation of a standardized solution means that modules can be taken and ported to other applications easily and used in other areas such as web and mobile applications and effectively addressing half of the issues from the OWASP current top ten list without relying on individuals to take in consideration security issues one by one.

7 References

[1] "eXtensible Access Control Markup Language (XACML) Version 3.0," 22 January 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. [Accessed 4 November 2014].

[2] "AT&T XACML 3.0 Implementation," [Online]. Available: https://github.com/att/XACML. [Accessed February 2015].

[3] A. I. G. M. Luigi Atzori, "The Internet of Things: A survey," *Computer Networks 54,* p. 2787–2805, 14 June 2010.

[4] N. B. A. C. V. M. Z. Andrea Zanella, "Internet of Things for Smart Cities," 2014 April 2014. [Online]. Available: http://eprints.networks.imdea.org/id/eprint/740. [Accessed 12 February 2015].

[5] A. V. V. J. W. J. L. D. Q. Qi Jing, "Security of the Internet of Things: perspectives and challenges," *Wireless Netw,* p. 20:2481–2501, 17 June 2014).

[6] J. W. C. Z. J. L. Hui Suo, "Security in the Internet of Things: A Review," in *International Conference on Computer Science and Electronics Engineering*, 2012.

[7] A. R. L. G. A. C.-P. S. Sicari, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks 76,* p. 146–164, 15 July 2014.

[8] Y. S. J. W. Prasant Misra, "Towards a Practical Architecture for the Next Generation Internet of Things," 3 February 2015. [Online]. Available: http://arxiv.org/pdf/1502.00797v1.pdf. [Accessed 10 Februry 2015].

[9] "Wikipedia M2M," Wikimedia Foundation, Inc., 13 February 2015. [Online]. Available: http://en.wikipedia.org/wiki/Machine_to_machine. [Accessed 16 February 2015.].

[10] S. M. I. J. W. M. I. S. G. M. I. X. L. M. I. a. V. C. L. F. I. Min Chen, "A Survey of Recent Developments in Home M2M Networks," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 16, NO. 1,* pp. 98-114, 2014.

[11] X. Z. Xiao Nie, "M2M Security Threat and Security Mechanism Research," in *3rd International Conference on Computer Science and Network Technology*, 2013.

[12] M. A. P. O. S. N. M. L. t. H. David S. Watson, "Machine to Machine (M2M) Technology in Demand Responsive Commercial Buildings," in *2004 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, CA, 2004.

[13] S. P. S. V. K. Roshni Bajpayee, “Big Data: A Brief investigation on NoSQL Databases,” *International Journal of Innovations & Advancement in Computer Science, IJIACS, Volume 4, Issue 1,* p. 2347 – 8616, January 2015.

[14] P. P. V. K. Rajendra Kumar Shukla, “Big Data Frameworks: At a Glance,” *International Journal of Innovations & Advancement in Computer Science, IJIACS, Volume 4, Issue 1,* p. 2347 – 8616, January 2015.

[15] “Google Images,” [Online]. Available: http://1.bp.blogspot.com/-4sM6wRgPBUA/T9FiKnMhNFI/AAAAAAAAACc/bOO8PGQ0rDs/s1600/BigData+-+V5+Lens.JPG. [Accessed 17 February 2015].

[16] G. Lafuente, “The big data security challenge,” *Network Security,* January 2015.

[17] “Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage,” *computer methods and programs in biomedicine 110,* p. 99–109, 2013.

[18] S. A. H. A B M Moniruzzaman, “NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison,” *International Journal of Database Theory and Application,* 2013.

[19] “MongoDB HomePage,” MongoDB, Inc., [Online]. Available: http://www.mongodb.org/. [Accessed 7 February 2015].

[20] “Apache Hadoop,” The Apache Software Foundation, 12 12 2014. [Online]. Available: http://hadoop.apache.org/. [Accessed 7 February 2015].

[21] “Apache Hbase,” The Apache Software Foundation, 11 Februaray 2015. [Online]. Available: http://hbase.apache.org/book.html. [Accessed 15 February 2015].

[22] “Apache Hive,” The Apache Software Foundation, 12 January 2015. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/Home. [Accessed 7 February 2015].

[23] “Apache Spark,” The Apache Software Foundation, [Online]. Available: http://spark.apache.org/. [Accessed 7 February 2015].

[24] “Apache Spark summary,” 2015 Black Duck Software, Inc., January 2015. [Online]. Available: https://www.openhub.net/p/apache-spark. [Accessed 16 February 2015.].

[25] “Apache Cassandra,” The Apache Software Foundation, [Online]. Available: http://cassandra.apache.org/. [Accessed 16 February 2015].

[26] “Wikipedia MongoDB,” Wikimedia Foundation, Inc, 9 March 2015. [Online]. Available: http://en.wikipedia.org/wiki/MongoDB. [Accessed 12 March 2015].

[27] “Tutorialspoint Neo4j,” [Online]. Available: http://www.tutorialspoint.com/neo4j/neo4j_features_advantages.htm. [Accessed 18 February 2015].

[28] “Neo4j official,” Neo Technology Inc., 2015. [Online]. Available: http://neo4j.com/. [Accessed 18 February 2015].

[29] “OrientDB official site,” Orient Technologies, 2015. [Online]. Available: http://www.orientechnologies.com/orientdb/. [Accessed 18 February 2015].

[30] “Redis website FAQ,” Salvatore Sanfilippo and Pieter Noordhuis, [Online]. Available: http://redis.io/topics/faq. [Accessed 20 February 2015].

[31] “Redis official website,” Redis, [Online]. Available: http://redis.io/. [Accessed 20 February 2015].

[32] R. M. M. A. S. Anam Zahid, “Security of Sharded NoSQL Databases:,” in *Conference on Information Assurance and Cyber Security (CIACS),* 2014.

[33] C. P. N. Priya P. Sharma, “Securing Big Data Hadoop: A Review of Security, Issues, Threats and Solution,” *International Journal of Computer Science and Information Technologies, Vol. 5 (2),* pp. 2126-2131, 2014.

[34] N. G.-O. Y. G. E. G. J. A. Lior Okman, “Security Issues in NoSQL Databases,” *International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11,* pp. 541-547, 2011.

[35] “A comparison between several NoSQL databases with comments and notes”.

[36] M. G. M. I. Enrico Barbierato, “Performance evaluation of NoSQL big-data applications using multi-formalism models,” *Future Generation Computer Systems 37,* p. 345–353, 2014.

[37] D. CORPORATION, “Benchmarking Top NoSQL Databases, A Performance Comparison for Architects and IT Managers,” 2013.

[38] “XML wikipedia,” Wikimedia Foundation, Inc, March March 2015. [Online]. Available: http://en.wikipedia.org/wiki/XML. [Accessed 20 March 2015].

[39] “JSON org,” [Online]. Available: http://json.org/. [Accessed 20 March 2015].

[40] “OASIS official wabsite,” OASIS, 2015. [Online]. Available: https://www.oasis-open.org/org. [Accessed 20 March 2015].

[41] “XACML wikipedia,” Wikimedia Foundation, Inc., 16 January 2015. [Online]. Available: http://en.wikipedia.org/wiki/XACML. [Accessed 20 March 2015].

[42] “JSON Profile of XACML 3.0 Version 1.0,” 15 May 2014. [Online]. Available: http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd03/xacml-json-http-v1.0-csprd03.pdf. [Accessed 11 February

2015].

[43] “SMARTIE Homepage,” IHP GmbH, 22 July 2014. [Online]. Available: http://www.smartie-project.eu/project.html. [Accessed 20 February 2015.].

[44] R. B. S. M. M. P. Jayavardhana Gubbi, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems 29,* p. 1645–1660, 24 February 2013..

[45] “Apache Hadoop YARN,” Apache Software Foundation, 11 13 2014. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. [Accessed 7 February 2015].

[46] “Apache Hadoop HDFS,” Apache Software Foundation, 13 11 2014. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. [Accessed 6 February 2015].

[47] “Apache Hadoop MapReduce,” Apache Software Foundation, 13 11 2014. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. [Accessed 7 February 2015].

[48] Y. H. A. C. A. G. G. H. E. N. Hanson, “Major Technical Advancements in Apache Hive,” in *SIGMOD '14 Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2014 .

[49] M. C. M. J. F. S. S. I. S. Matei Zaharia, “Spark: Cluster Computing with Working Sets,” 2010.

[50] M. C. F. X. D. L. a. K. Z. Jiafu Wan, “From Machine-to-Machine Communications towards Cyber-Physical Systems,” *Computer Science and Information Systems Vol. 10, No. 3,* p. 1105–1128, June 2013.

[51] C. S. DU Jiang, “A Study of Information Security for M2M of lOT,” in *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE),* 2010.

[52] “Wikipedia,” Wikimedia Foundation, Inc., 13 February 2015. [Online]. Available: http://en.wikipedia.org/wiki/Machine_to_machine. [Accessed 16 February 2015.].

[53] “XML website,” O’Reilly Media, Inc., [Online]. Available: http://www.xml.com/. [Accessed 20 Merch 2015].