University of Aveiro
2015.

Department of Electronics, Telecommunications and Informatics

Vedran Semenski

An ABAC framework for IoT applications, utilizing the OASIS XACML standard

This thesis is submitted to the University of Aveiro and Faculty of Electrical Engineering and Computing, University of Zagreb for compliance with the requirements for the degree of Master of Science in Computing, performed under the supervision of prof. dr. sc. Óscar Pereira, Professor on the Department of Electronics, Telecommunications and Informatics of the University of Aveiro and Mr. Sc. Ricardo Azevedo, from PT – Inovação e Sistemas

I dedicate this work to my loving parents without whom, this would never have been possible.

The jury

Presidente	Prof. Dr. André Ventura da Cruz Marnoto Zúquete, Professor Auxiliar da Universidade de Aveiro
Arguente	Prof. Dr. Miguel Filipe Leitão Pardal Professor Auxiliar da Universidade de Lisboa
Orientador	Prof. Dr. Óscar Mortágua Pereira Professor Auxiliar da Universidade de Aveiro

acknowledgements

I take this chance to extend my thanks to all the people who helped and supported me trough my studies and during this work.
To prof. dr. sc. Óscar Pereira, my supervisor, and mr. sc. Ricardo Azevodo, my co-supervisor, for giving me clear guidance and help when it was needed.
To my professors from the courses I took while studying on FER, University of Zagreb and the last year on DETI, University of Aveiro, for providing me with the opportunity to learn and giving direction.
Lastly, I especially want to thank my family for their support and encouragement during the years of my studies.

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

resumo IoT (*Internet of Things*) é uma área com futuro promissor, que apesar dos seus problemas já terem soluções satisfatórias, a segurança permaneceu de certa forma esquecida, continuando a ser um grande problema. Controlo de acesso é uma forma de reforçar segurança que envolve avaliar pedidos de acesso a recursos e negar o acesso caso este não seja autorizado, providenciando assim segurança para recursos vulneráveis. Controlo de Acesso é um termo lato e que consiste em diversas metodologias, das quais as mais significantes são: IBAC (*Identity Based Access Control*), RBAC (*Role Based Access Control*) and ABAC (*Attribute Based Access Control*). Neste trabalho vai ser usado ABAC, já que oferece uma maior flexibilidade comparativamente a IBAC e RBAC, requerendo menos requisitos de manutenção e longevidade pela sua natureza adaptativa. OASIS (*Organization for the Advancement of Structured Information Standards*) desenvolveu XACML (*eXtensible Access Control Markup Language*), um padrão para escrita/definição de pedidos/políticas e de avaliação de pedidos sobre conjuntos de políticas com o propósito de reforçar o controlo de acesso sobre recursos. XACML foi definido com o propósito de que os pedidos e as políticas fossem fáceis de ler por humanos, retendo uma estrutura bem definida que permita uma avaliação precisa. O padrão usa ABAC. Este trabalho tem o objetivo de criar uma estrutura de segurança que utilize os padrões ABAC e XACML para que possa ser usado por outros sistemas e reforce o controlo de acesso sobre recursos que careçam de proteção, garantindo acesso apenas a pessoas autorizadas. Vai também permitir uma definição pormenorizada de regras e de pedidos para permitir uma avaliação de maior precisão e com um maior nível de segurança. Os casos de uso principais são aplicações IoT, como aplicações Smart City, que inclui monitorização de trafego inteligente, energia e consumo de utilidades, monitorização pessoal de saúde, etc. Estas aplicações lidam com grandes quantidades de informação (*Big Data*) confidencial e/ou pessoal. Existe um número significativo de soluções NoSQL para resolver o problema, mas a segurança é ainda uma questão por resolver. Este trabalho vai usar duas bases de dados NoSQL. Uma base de dados *key-value* (Redis) para armazenamento de políticas e uma base de dados *wide-column* (Cassandra) para armazenamento de informação de sensores e informação de atributos adicionais durante os testes.

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 way of enforcing security that involves evaluating requests for accessing resources and denies access if it is unauthorised, therefore providing security for vulnerable resources. 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 and longevity. 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 by 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 of 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, etc. These applications 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
CRUD	Create, Read, Update, Delete
DAC	Discretionary Access Control
DoS	Denial of Service
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

1 Introduction

Security systems are often overlooked in growing and fast developing areas. It often comes last but is a necessary component needed for building systems working in a "real world" environment without compromising safety and confidential information. Tendencies and advancements in sensor technology are one of the main factors that gave birth and pushed advancements in IoT (The Internet of Things).

IoT offers great opportunity and possible applications that could and are changing everyday life but this comes with many problems and challenges. M2M (Machine to Machine) is a broad are that deals with connection and communication issues between constrained devices and because of the importance in sensor networks it is essential to IoT. Considering the vast amounts of data/information being generated by sensor networks, storage and processing issues fall in the area of Big Data. NoSQL (Not only SQL) database systems that address some of the challenges with Big Data. 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 is an issue that needs to be addressed in order to make the technology safe and will be the focus of this dissertation. Smart City projects are one of the biggest and most significant implementation scenarios in IoT. Implementations include smart traffic control, energy consumption monitoring and management, health monitoring and others. Data generated for these application are usually sensor networks, people's smart phones or other data sources like healthcare and education. The information used is therefore often private or confidential, and systems and services critical. The control over actuators, sensors and data can be misused if access is given to unauthorised subjects. Because of these facts, they must offer greater levels of security and need to implement long term and flexible solutions. 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 (Organization for the Advancement of Structured Information Standards) XACML (eXtensible Access Control Markup Language) standard. It will utilize one NoSQL(Not only SQL) database (Redis) for storing policies and another (Cassandra) for storing resources, in this case sensor data. The functionality of a PDP (Policy Decision Point) engine and configurable PIP (Policy Information Point) engine will be used from the AT&T XACML implementation [1] open source project The use case that will be considered as the end point for

integration are IoT applications and more specifically SMARTIE (Smart City), a smart city project founded by the EU with PT – Inovação e Sistemas as one of the partners.

Section 1 is divided as follows: subsection 1.1 contains the official short description text along with the objectives, subsection 0 will present the problem addressed by this work, in subsection 1.3 a short description of the proposed solution will be given and subsection 0 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 smart city 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, JSON will also be used.

Objectives:

This dissertation is focused on two main development activities: security framework and databases. 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 and Access Control is one way of dealing with this. Role Based or Identity Based Access Control is the way access control is commonly enforced. The relation between roles and sets of allowed actions over resources are intuitive to comprehend but common issue is the explosion of the number of roles [2] and the managing aspect of maintaining that type of system. Also, there is the issue of somewhat limited possibilities on defining conditions and actions over resources, therefore they do not offer much flexibility without considerable modifications or integrating complex automated processes in the implementations.

The OASIS's XACML standard 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 other areas. 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 a JSON variant was also created and SMARTIE will utilize that variant. This dissertation should therefore also focus on the JSON variant. The JSON variant is more memory efficient and easier to read in raw text format. The XACML v3.0 standard has been available since January 2013. but a complete and flexible implementation of has still not been made open source or available in other ways.

This dissertation will explore the areas of IoT, M2M (Machine to machine), 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 database for storing sensor data, creation of a security framework for enforcing access control that utilizes the OASIS XACML standard and the accompanying NoSQL database 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*

The solution utilizes a modified version of the architecture from the XACML standard [3] described in subsection 2.1.2, and two proposed general methods for implementation/integration by other systems described in subsection 4.3. Because of the qualities inherited from ABAC and XACML, it offers great flexibility for a number of possible use cases although the primary use case are IoT applications. The implementation and final solution are explained in more detail later in the document, in section 0. The enforcement of access control by other systems using this solution is done by implementing a PEP (Policy Enforcement Point) component, using a PAP (Policy Administration Point) Web App for creating and uploading of XACML policies, creating of XACML requests. The enforcement is done by sending requests to the PEP which forwards it for evaluation to the core system. The system returns the evaluation result and executes the action in the case of a positive result and terminates the request in case of a negative result. The solution doesn't implement any advanced actions and obligations defined by the XACML standard [3] and i it differentiates request results into positive and negative and acts as mentioned before.

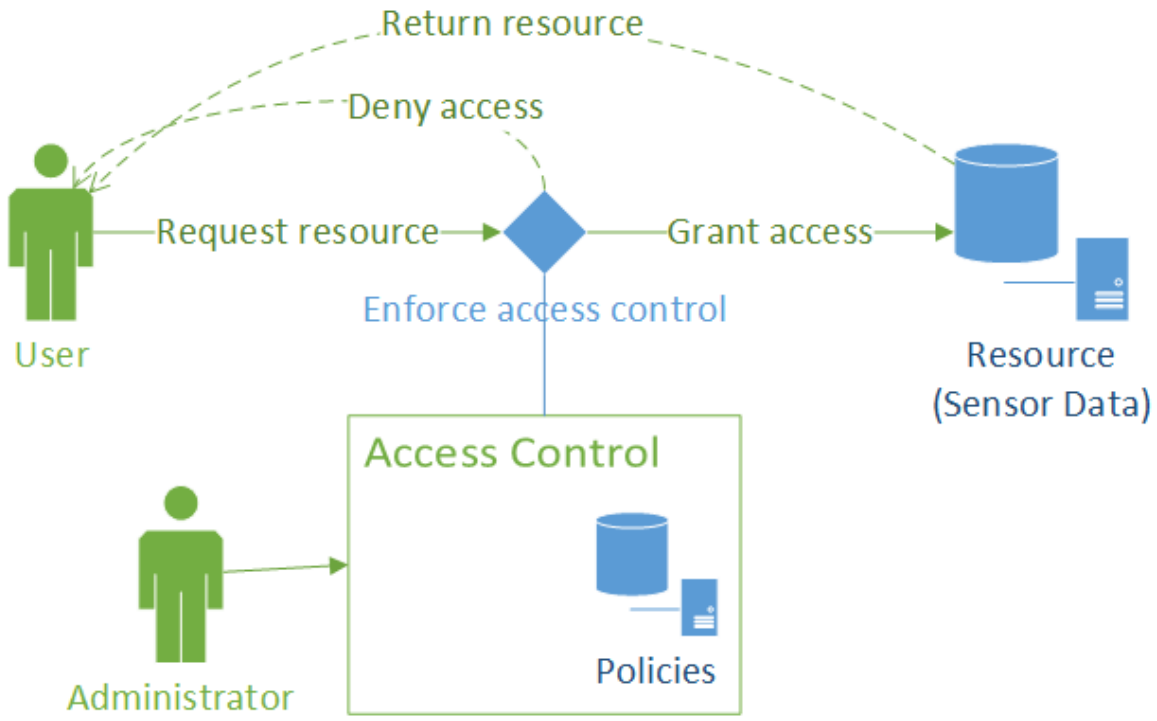

Figure 1. High - level overview of the architecture

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 framework offers a simple and fast way of integrating access control in an existing application by providing 2 main entry points. An administration point (PAP)

that is used for managing (uploading and deleting) 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 rules 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 with subjects, resources or environmental conditions (example: day of the week). As a result of this the target system is less prone to errors over time. The critical part from an implementation perspective is having well and correctly defined policies and making requests that are an accurate representation of the actual requests.

The solution allows for 2 ways of integrating into other systems. The first possibility is putting and running the whole 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.

1.4 *Development environment and technologies used*

The solution was built mainly in JAVA, using the Eclipse Luna IDE (Integrated Development Environment) for organizing all of the projects and code. Components from an open source project (AT&T XACML 3.0 implementation [1]) used for the XACML functionality it provides. Two NoSQL databases (Cassandra[4] and Redis[5]) and their Java clients. Additionally MySQL[6] with XAMPP[7] and a PostgreSQL[8] databases were used for initial testing purposes. The Axiomatics Alfa[9] plug-in for Eclipse was used to help create policies in the initial stages of development. Jersey[10] and JAX-RS[11] libraries were used for creating the REST service and clients. Lastly Maven[12] was used for solving dependencies and building issues when using multiple projects.

1.5 *Dissertation structure*

The structure of this dissertation is organized as follows: In section 2 present the current State of the Art along with a critical overview and presenting some related work, Section 3 will present background fields and related work, Section 4 contains the description of the proposed and developed solution, Section 5 presents test results in various test scenario, proof of concept and discussion for the integration in SMARTIE and other IoT applications, Section 6 contains the final conclusion and presents potential future work.

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. Firstly in subsection 2.1, as it is the core area of this dissertation, Access Control in general will be presented. It contains descriptions of the most relevant methodologies and will explain the significance each of along with giving a and brief comparison. Lastly, the OASIS XACML standard will be presented with some examples of policies and request in both XML and JSON and the proposed architecture. Subsection 2.2 will contain a description of IoT (Internet of Things) which is a broad term and is significant for this work as the main use cases targeted by solution are IoT applications. Subsection 2.3 will briefly explain M2M (Machine to Machine) because of its importance to sensor networks and therefore IoT. Subsection 2.4 will present the concept of Big Data and its relevance to IoT. Subsection 2.5 will present NoSQL (NoSQL) as a concept, present a list of currently most relevant NoSQL databases, covering each type, and present a performance and characteristics comparison overview with figures taken from other sources.

2.1 *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. Before continuing, two key terms need to be explained as they will be used a lot in this work:

1. **Subject** - this is a term used for the entity that is trying to access a certain resource. This can be a person, but it can also be a process, machine or any other computer system trying to access a resource;
2. **Resource/Object** - these two terms will be both be used and they represent anything that access control is being enforced upon. This means that a resource can be data from a database, access to a application, service, access to sensors, actuators, facility (room, building), actions over resources etc. This wide definition is needed because of the wide variety of use-cases.

Access control is a security technique that enforces security over resources by limiting access to them. The access is given only to authorised subjects which can be people or other systems, depending on the implementation. A typical workflow with access control would consist

of: receiving a request for a certain resource, evaluation of the request against one re more policies and allowing or denying the request depending on the evaluation result. The systems has to have an architecture to facilitate enforcement of access control, a evaluation methodology and defined policies (or rules) for evaluating the requests. The significance, complexity and size of these of course vary from implementation to implementation and can depend heavily on the business layer of the system that is integrating access control.

2.1.1 Types of access control

In this section a few of the most important and significant types of access control will be presented. Examples will be shown and a comparisons will be given while presenting each type of Access Control.

DAC (Discretionary Access Control) and MAC (Mandatory Access Control)

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).

File	Owner	Permission for others
X	Adam	Read
Y	Barry	Read,Write
Z	Charley	None

User\File	X	Y	Z
Adam	Read,Write	Read,Write	None
Barry	Read	Read,Write	None
Charley	Read	Read,Write	Read,Write

Table 1. DAC example: file ownership relation table with permissions (left), resulting matrix with allowed actions for every user(right)

In Table 1 and example tables which could be used for managing access and actions over files. The users that are owners (and usually creators) of files have all authority over the files and can change the allowed actions for other users. If users try to execute actions over files they don't have permission for, their request will be denied. The table on the left, contains the column *permission for others* which can be edited only by the owner. The table on the right shows the resulting set of allowed actions for all users over all of the files used in this example.

MAC on the other hand is a type of access control where an external entity like an administrator decides on giving or denying access to resources. This means that individual subjects

cannot change the policies and grant access rights to itself or other subjects [13][14]. An 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 and power to give or remove permissions over them. The objects/resources are often divided into multiple categories according to the level of security or importance they fall into. This categorisation is often organized as a hierarchy. The categories could for example be: Critical, Semi-critical, Non-critical, or a simplified version would be Private and Public. Than the subjects would have credentials that would reflect the type of actions they can execute. These credentials are therefore permissions and the administrator can modify these along with changing the category an object/resource is assigned to.

Object\Subject	LOW,LOW	LOW, HIGH	HIGH, LOW	HIGH, HIGH
LOW, LOW	Allow, Allow	Allow, Allow	Allow, Allow	Allow, Allow
LOW, HIGH	Allow, Deny	Allow, Allow	Allow, Deny	Allow, Allow
HIGH, LOW	Deny, Allow	Deny, Allow	Allow, Allow	Allow, Allow
HIGH, HIGH	Deny, Deny	Deny, Allow	Allow, Deny	Allow, Allow

Table 2. Example of MAC permission matrix for READ/WRITE operations

In Table 2 an example matrix for MAC can be seen. The *HIGH* and *LOW* values represent the example security category levels for the objects and subjects for *Read* and *Write* operations. In this example it can be seen that objects can execute operations only if the level of that operation is equal or higher than the level specified for the Object. This allows for the administrator to make individual changes for certain resources or types of subjects but also to make global changes at one point that can affect all.

IBAC (Identity Based Access Control)

IBAC (Identity Based Access Control) is based on giving access to a resource depending on a users identity. An example of an IBAC implementation would be having a list of with authorised users and denying access to every unlisted and therefore unauthorised user. Implementations often become RBAC because identities become grouped, groups have different permissions or can access different resources and therefore they can be viewed as roles. IBAC can be very fine grained as it can have different permissions defined for every user but this has the downside of being very hard to manage and maintain. This also means that they are not very flexible. Most adequate implementations would be as mentioned before, or password (authentication card, fingerprint...) based authentication systems that limit access to a resource which could be an account, secure room, traffic service etc. [15]

RBAC (Role Based Access Control)

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.

UsedID	Role
1	Admin
2	User
3	User
4	User
5	User
6	User

Role	Permissions
Admin	C, R, U, D
User	C, R, U
Visitor	R

Table 3. RBAC example: table containing users and their assigned roles (left), table containing list of permitted actions for each role

Table 3 shows an example of possible types of roles that can be found in a simple web site scenario. In the right table the C, R, U and D are taken from the CRUD (Create, Read, Update, Delete) acronym, often used in web site terminology. It shows that a user with the role *Admin* can execute all of the possible actions meaning he has authority over all of the content available on the web site. The *User* role doesn't have permission for executing a *DELETE* action. This means that any delete action that is maybe presented to the user like "delete trash" or "delete image" has to either be authorised by the admin or is just becomes hidden, but not deleted. Still, the *User* role can *CREATE* and *UPDATE* content. The *User* role is adequate for registered users. The *Visitor* role has only *READ* permissions therefore it can only view content and not create new content. The visitor role is therefore adequate for all unregistered visitors to a website allowing them to view but not to create of change content before they register or sign in. The table on the left shows a simple

list of 6 registered users, of which one is the administrator. Visitors are not registered user so they do not appear in this table.

ABAC (Attribute Based Access Control)

ABAC (Attribute Based Access Control) is a type of access control that evaluates requests against policies according to attribute values. Attributes are typically divided into three categories:

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

These attributes therefore contain data from the subject trying to access the resource, data from the resource that is being accessed and environmental data which represent current conditions. When a request is being evaluated, the decision is made according to these values and conditions/rules defined in policies. Policies are commonly defined in policy files and contain rules and conditions for evaluation. An example policy would be restricting people under the legal age limit to apply for a driving licence. The evaluation would require getting the subjects age (or date of birth) and the legal age limit which would be an environmental attribute. Only after getting these attributes, comparing them and confirming that the subject is over the age limit, the request can be allowed. Policies can, of course, also work on a deny principle and define conditions/rules that would deny access to certain resources. Keeping close to the previous example, one with a deny principle would be defining a policy that would deny access to driving a car (by blocking the car's accelerator pedal) if it is detected that a person has a high alcohol level. Policies therefore contain rules and conditions that have to be defined and met for a decision to be made after comparing all the attributes needed for evaluation.

id	name	date of birth
1	Person1	1.1.1980.
2	Person2	1.1.2000.

action	age limit
get driving licence	18
vote	18

Table 4. ABAC attributes example: table containing subjects attributes (left), table containing environment attributes (right)

Table 4 shows an example described before. If the current year is 2015., which is also a environmental attribute, *Person1* can both vote and get a driving licence but *Person2* cannot. A policy is that specific condition that defines conditions when access can be given or denied. Another example would be a policy that states that a user can add or change data related only to accounts that they created and are therefore owners of. This policy would therefore stop any unwanted changing of accounts from other users as access would not be allowed.

Because of these attributes RBAC can be viewed as a subset of ABAC because an ABAC system that has policies relying on a single attribute containing the subject's role would be equivalent to a RBAC system. ABAC can therefore be made compliant and integrated in place of any RBAC system and would offer more possibilities and easier use of more complex conditions and rules. Examples like the one shown in Table 3 can also work with ABAC simply by evaluating everything based the attribute that contains a subjects 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 as 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.

2.1.2 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[16].

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 [17].

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 [18].

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 thought 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 [19][20]. 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 [3][21]. 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 [22] 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 the age limit (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 the age limit</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 2. Example of a XACML policy

In Figure 2 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 3. Example of a XACML request in the JSON format

In Figure 3 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.

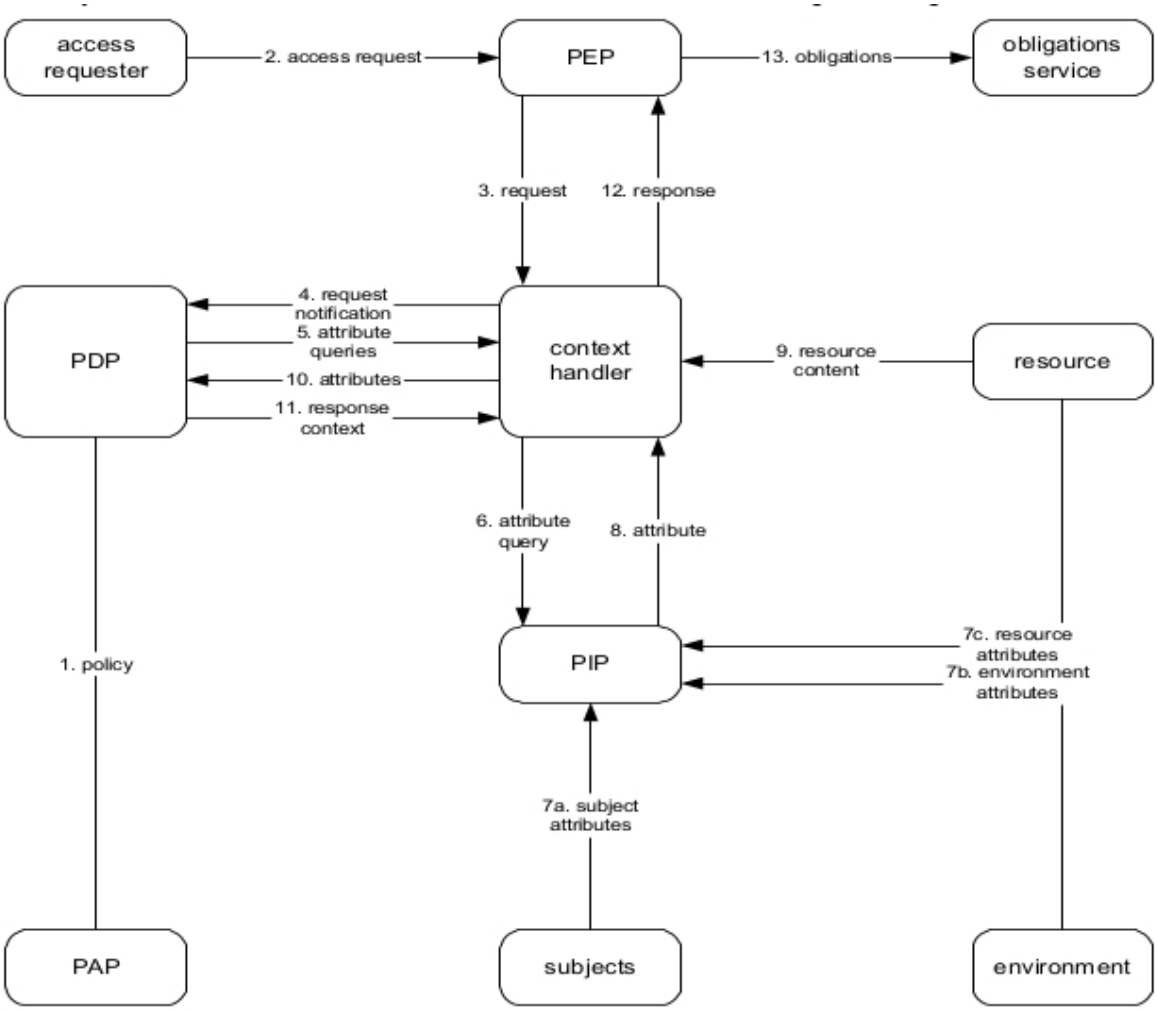

Figure 4. Architecture proposed by the OASIS XACML standard [3]

Figure 4 shows the architecture proposed in the OASIS XACML standard. The components are as follows:

- PEP (Policy Enforcement point) - component that performs access control by performing the decision provided by the response. This may also mean fulfilling obligations that come in the response;
- PDP (Policy Decision Point) - this component is responsible for evaluating the request against a policy. It contains all the functionality to make the evaluation and produce a response;
- PIP (Policy Information point) - This component is responsible for retrieving attributes. The attributes are split into three types: subject, environment and resource attributes;
- PAP (Policy Administration Point) - the policy administration point contains the functionality required for managing policies. Typically this means adding and removing policies;

- access requester - entity that is requesting a resource;
- obligations service - service that executes any obligations after the evaluation is complete
- resource - entity containing one or more resources and resource attributes that the *access requester* is trying to access
- subjects - entity containing subject attributes. Typically the subject attributes are attributes of the *access requester*.
- environment - entity containing one or more environmental attributes.

One other component that should be mentioned as it was part of the architecture in version 1.0 of the standard and will be relevant for the work described later in this dissertation is the PRP.

- PRP (Policy Retrieval Point) - component used for retrieving of policies. In version 1.0 it is used to provide the PDP with policies unlike in version 3.0 as shown in Figure 4 which uses the PAP for that purpose

The removal of a PRP from the new architecture and having that functionality integrated in to the PAP could bring issues. Because of this, the final solution has an architecture that is different from and will be addressed in section 4.

2.2 *The Internet of Things (IoT)*

In this subsection 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[23]. 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 [24][25]. 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 [24] wrote.

They explain that the two main views that come from the name "Internet of Things" and one additional one that is a result from recent tendencies in IoT developments. 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. The third view, which the authors defined as a "Semantic oriented vision" 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. The authors of [26][27][28] describe and stress the importance of security for a widely acceptable solution. They divided IoT 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 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 utilize a number of older and newer technologies, the growing number of networks and devices bring up more issues with security, data integrity and scalability. Standardisation of some aspects regarding communications and security enforcement could solve many of these problems but brings with itself the issue of wide acceptance and implementation.

A more future oriented view and analysis is provided by [29][30]. 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 over 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. Additionally it contains a 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 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 that 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 [28][26] give a good overview of the current situation in IoT and a give proposals for more uniformed future research and developments.

2.3 *Machine to Machine communication (M2M)*

This section will provide a brief overview of the current state, issues and a future developments in M2M (Machine to machine). M2M 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 [31].

The motivation behind M2M comes as stated in [32] mainly comes from three 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 [32] [33][34]. M2M communication can be achieved with a number of technologies but the most significant and most promising ones are wireless technologies. In [32] the main technologies that are mentioned and reviewed 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. Trends in M2M as mentioned in [32] 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 dealt with regarding M2M.

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

1) Terminal layer - contains the access gateway, 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. 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. This was, of course, only one simple example of a potential attack scenario among many other possible ones.

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.

2.4 *Big Data*

Storing, processing, accessing and managing vast amounts of data with unreliable or complex structures 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. Same as M2M, Big Data is often referred to together with IoT as IoT applications typically generate and need to process a large amount of data. These challenges are addressed by Big Data.

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 [35][36] and can be seen in Figure 5.

Figure 5. The 5 Vs of Big Data[37]

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 [35] 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 [35] 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 [38] 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.

2.5 *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[39][35]. 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. 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 (Relational Database Management System) systems. RDBMS systems usually have ACID (Relational Database Management System) characteristics.

ACID characteristics stand for:

- Atomicity - ensures that a transaction and all of its actions and consequences will execute either in its entirety or fail. This therefore means that transactions cannot be split or executed partially;
- Consistency - ensures that a database has to be in a valid state before and after a transaction executes, but it doesn't ensure the correctness of the transaction.
- Isolation - ensures that the result of executing transactions concurrently, is equal to the result when executing the same transactions serially. This also needs to persist even in the case of transactions failing;
- Durability -this ensures that for every transaction that has been committed, the results of it will still persist, even in the case of system failures. For example: crashes, errors, power losses etc.

BASE (Basically Available, Soft-state, Eventually consistent) characteristics, on the other hand do not force and guarantee consistency. It is somewhat an optimistic view and accepts that the database is "eventually consistent". This loose characteristics can be beneficial for replication and partition characteristics, therefore is suited for massively distributed systems.

Because RDBMS systems 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 [40], and because of this most systems can be described as BASE.

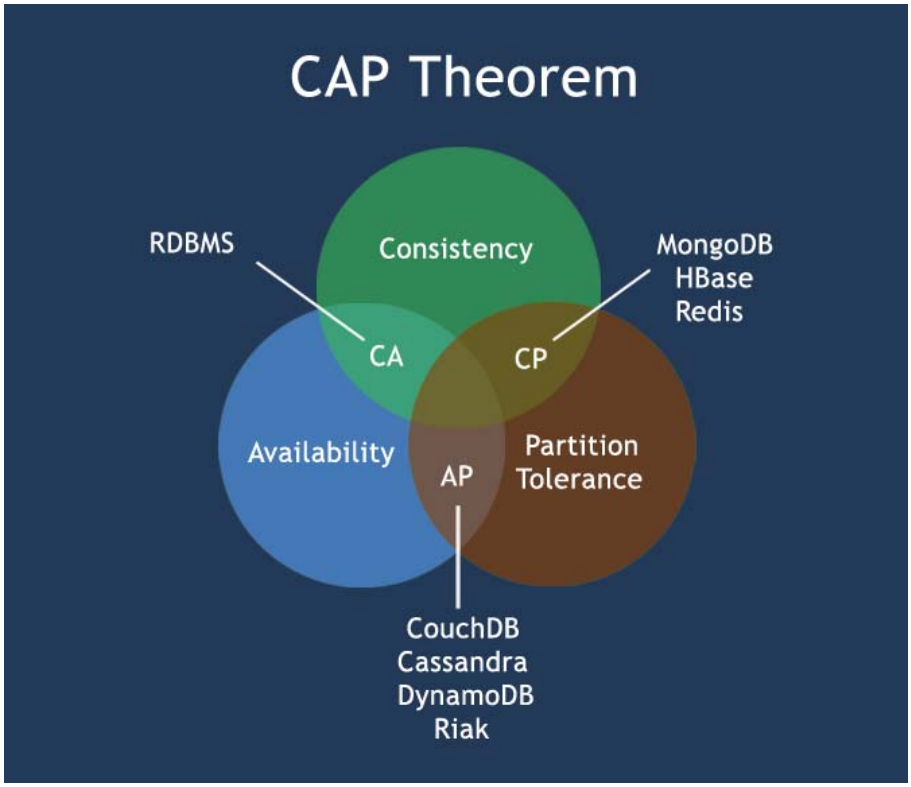

Table 5. CAP theorem diagram

The CAP diagram, also known as "Brewer's theorem", describes the relation between Consistency, Availability and Partition tolerance characteristics. It shows that it is impossible to have all of the characteristics without significantly sacrificing one. Table 5. shows this and also gives examples of databases and where they fall into. RDBMS systems are not appropriate for Big Data applications because of the lack of partition tolerance. NoSQL typically have partition tolerance but then differ from one another regarding Availability and Consistency characteristics.

Hadoop

Hadoop is an important platform for NoSQL as many of the most popular NoSQL databases are built on top of it as it. Because of this many of these solutions are compatible with one another and therefore offer a large and very powerful ecosystem of databases and data processing solutions. Hadoop is an Apache project and was created as a platform for developing open-source software for solving scalability and processing problems in the context of large quantities of data. Hadoop is mainly a file storage system that is designed to store large amounts of data, distributed over large clusters, reliably and stream data to those who need it with high speed and efficiently. The work from [41] mentioned that in 2010. Yahoo! used it to manage 25 petabytes of data. 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"[42]. 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 its official webpage[42]:

- 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 (Yet Another Resource Negotiator): A framework for job scheduling and cluster resource management;
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

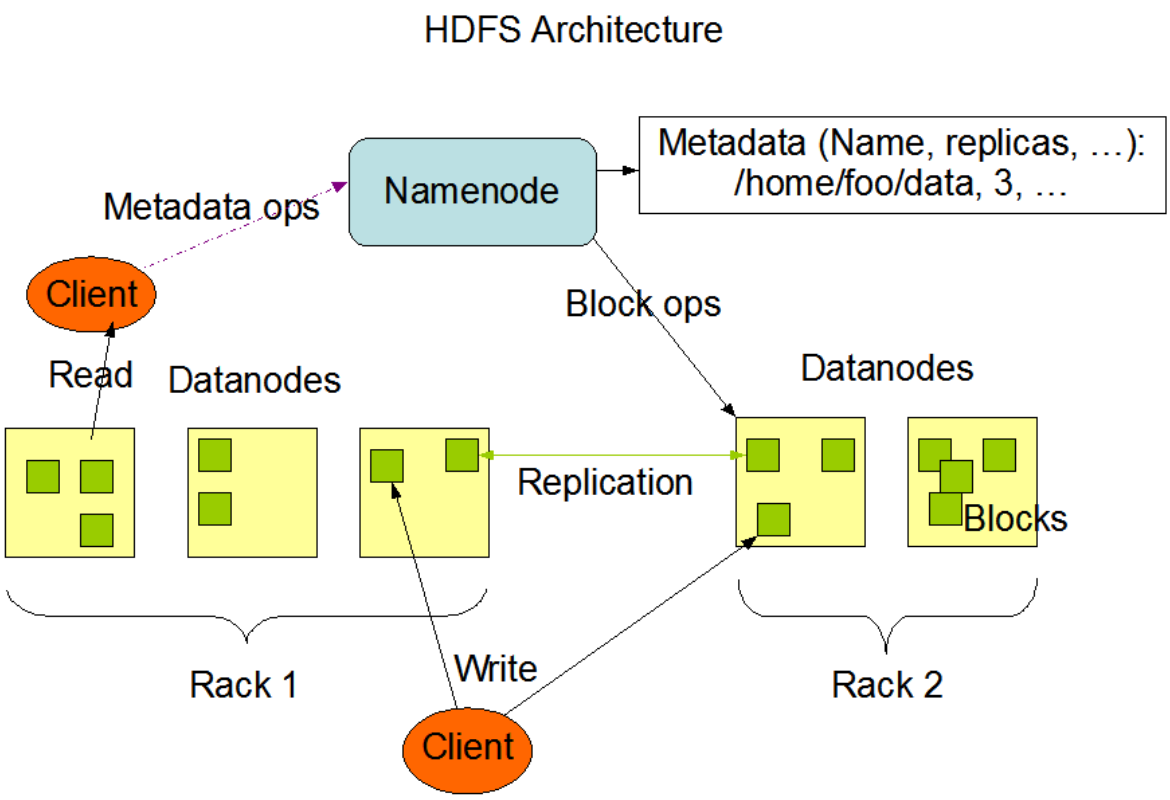

Figure 6. Hadoop HDFS Architecture [43]

Figure 6 shows the architecture of an HDFS cluster. It is a master/slave architecture and consist of a single *NameNode* and a number of *DataNodes*. Both are pieces of software designed to run on commodity machines which typically run a Linux operating system. The *NameNode* is a

master server responsible for managing the namespace and regulates access to files by clients. It determines mapping options for the *DataNodes* and it can execute file system namespace operations. *DataNodes* are responsible for managing data storage attached to the nodes they are running on. They perform actions over blocks and are responsible for serving read and write operations for the systems clients. The system is greatly simplified because of the existence of a single *NameNode* that manages the namespace it can contain all the metadata for that namespace.

Hadoop MapReduce is a software framework that allows for writing applications which need to process large amounts of data parallel on large clusters. The amount of data can be multi-terabyte data-sets and it can run n thousands of nodes. On Hadoop MapReduce the nodes responsible for computing a set of data are typically the same node where the data is located making the execution much simpler and faster[44]. The user making MapReduce jobs typically specifies the map and reduce functions and the underlying systems parallelizes the process across the nodes in an efficient manner. It handles failures and re executes failed tasks and coordinates operations for efficient network communication, disk and processing usage. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks[45].

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[46]. 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.

2.5.1 Types of NoSQL databases

NoSQL databases can be classified in four basic categories as done in [40].

1. Key-Value stores;
2. Document databases (or stores);
3. Wide-Column (or Column-Family) stores;
4. 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 to be semi-structured 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 semi-structured 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 [47] 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.

2.5.2 Current state of NoSQL technologies

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.

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 [48] 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[48] 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 [49] 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 updates, 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 [49]:

- 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 [50] 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 [51] 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 an 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[4]. 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 [47][52]:

- 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 [53][54]:

- 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 [55] 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 [55]:

- 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[56]. 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 [5]:

- 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.

2.5.3 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 mainly utilized for scalability, it bring to the surface a lot of security issues in need of dealing with.

The authors of [57][58] [59] 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. Table 6 shows the result of research done by [57].

NoSQL database / Assessment Criteria	MongoDB	Redis	CouchDB	Cassandra	HBase	Couchbase Server
Authentication	Medium	Low	Medium	Low	Medium	Medium
Access Control	High	Low	Low	Low	Medium	Low
Secure Configuration	Medium	Low	Low	Low	Low	Low
Data Encryption	Medium	Low	Medium	Medium	Low	Low
Auditing	Low	Low	Medium	Low	Medium	Medium

Table 6. Comparative analysis of sharding security in various NoSQL databases[57]

2.5.4 Performance studies and comparisons

Comparisons and performance evaluations given by the authors of [40][60][36][61] [62] 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.

The performance evaluation given by [62] can be seen in Figure 7, Figure 8, Figure 9, and Figure 10. This, along with the information found in other sources mentioned before in this subsection, was the basis behind choosing the databases for this solution. That decision is shown in subsection 4.1.1.

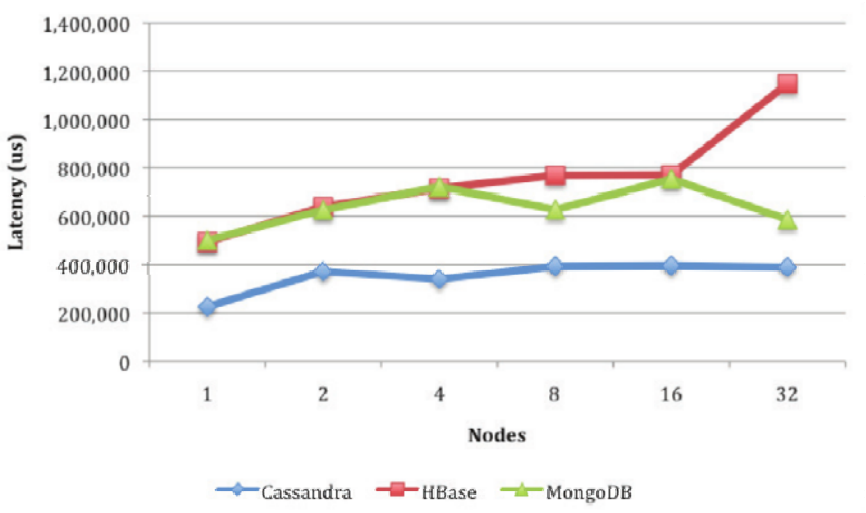

Figure 7. Read latency across all workloads[62]

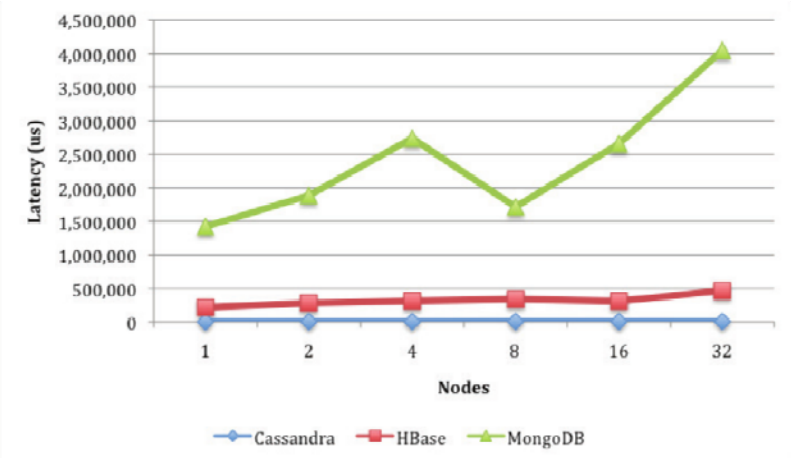

Figure 8. Insert latency across all workloads[62]

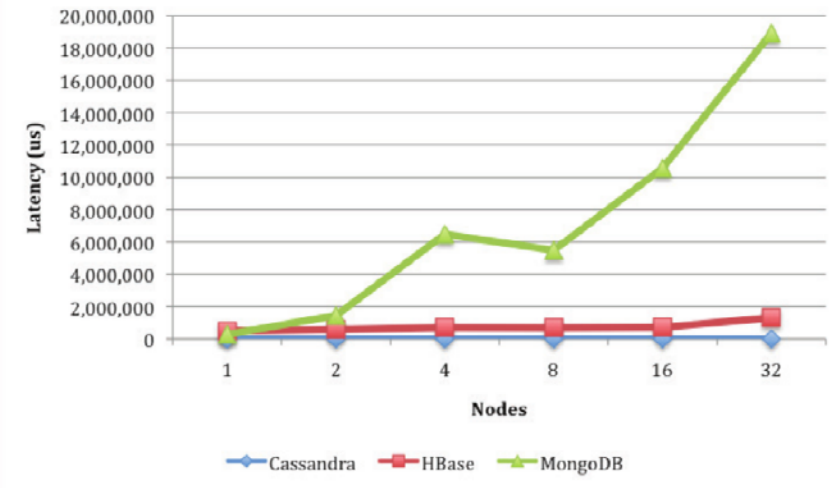

Figure 9. Update latency across all workloads[62]

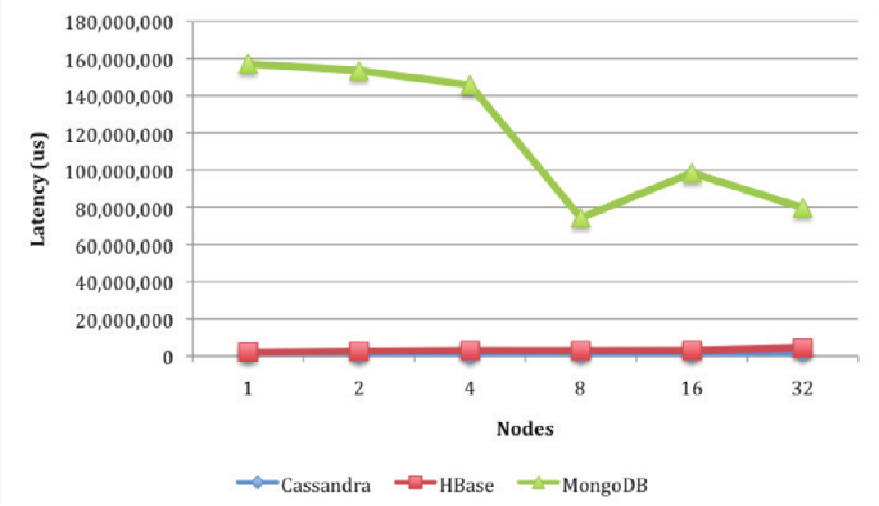

Figure 10. Scan latency across all workloads[62]

2.5.5 NoSQL systems summary

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.

3 Background

3.1 *Smart Cities*

As mentioned in[63], the term smart city is widely used, often outside of the computer science context but rather in a more social and cultural context. Definitions therefore vary and many exist but he final aim is to make a better use of the public resources, increasing the quality of the services offered to the citizens while reducing the operational costs of the public administrations. The context that is regarded to in this work is as an IoT application scenario. We will define it as:

"A city utilizing an infrastructure of sensor networks and services to collect and utilize the generated data for the main purpose of improving efficiency and managing of certain aspects of the city like: traffic, energy and utilities, healthcare, public safety, education etc.".

IoT, Big Data and security are therefore areas essential to smart cities. They are one of the applications that offer great promise in the improvement of everyday life and because of the infrastructure could also help in scientific research.

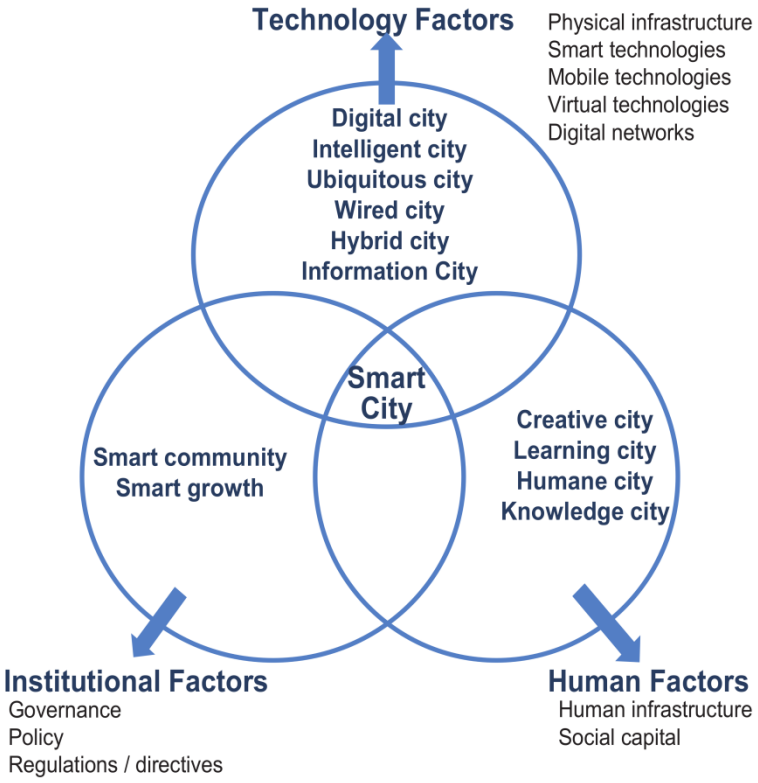

Figure 11. Fundamental components of a Smart City[63]

In Figure 11 the fundamental components of a Smart City as defined by [63] can be seen. This is a general and "outside" view rather than a technological one and it describes the factors that are needed for a Smart City to exist. These are three factors: Technology, Human and Institution. the Technology factor represents all of the technologies needed for a city to become "smart". On a minimum they would need to include a physical infrastructure and a software component for utilizing it. As the smart city is mainly cantered around benefits for the cities citizens the human element is essential. The degree of acceptance has to be high for the concept to work therefore the functionality has to compliment the needs of the citizens and needs to ensure safety of personal information. The institutional component incorporates the regulations and planning that has to take place. The government and regulatory bodies need to be fully included to utilize this technology as efficiently as possible. This king of large scale projects need plans for growth, regulations and managing solved which all needs to be governed by the city's authorities. Bureaucracy issues could delay projects like these for a significant period, therefore these issues need to be dealt with in a timely and efficient manner.

Smart cities are very promising projects as they offer improvements to everyday lives for their citizens, more efficient management of resources, science and others, all with a lower maintenance overhead than before. As they are large scale applications they have many issues and challenges to overcome. After they become a reality and are proved to work, adaptation to other areas will come easy, so cities and their citizens are not the only ones that could benefit.

3.1.1 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 [64] 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 €.

Vision [64]

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 life 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[64]

- 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 [64]

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 [64]

- 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 [64]

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

3.1.2 Other Smart City projects

These is a number of smart cities that exist and are currently taking place. Most are still in somewhat early stages and need to be fully built and proven to move from a research projects on universities to larger scale projects. Some aspects of Smart Cities are already visible in areas as traffic. Monitoring has become common and things like traffic sensors connected to a centralized systems have existed for some years. These systems maybe wouldn't fall in the category of "Smart" but aim towards the same goals as smart cities. A fully developed smart city solution would improve on these significantly and would need less maintenance.

Padova is one of the more significant IoT smart city pilot application that were done in recent years [65]. From the published work a general architecture can be seen that contains the sensor database, connections to the sensors for collecting data, an interfaces for direct sensor access and a web front-end interface. It was tested using lights sensors placed on a road from which the number of passing cars could be gathered and simple traffic monitoring could be realised. It is stated that it was built with expansion in mind so the progress from this project could prove to be significant in this area.

As mentioned, there are a other 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. This is a crucial part because the involvement of the EU means faster solving of any bureaucracy or other non technical issues along with possibly more founding in case of the project's success. 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. This is an important fact as practically, it means easier adaptation to different cities and their needs.

3.2 *Related work*

This section will present some projects, implementations, products that are related and relevant to the work done in this dissertation. A brief overview and explanation will be given along with a critical opinion for every one of these. These were a result of research done while doing this work and is possible that it's is not complete or completely accurate. Other significant solutions therefore could exist but did not come up during the research process.

Axiomatics

Axiomatics is Stockholm, Sweden based company that is currently leading in the area of implementations utilizing ABAC and XACM. They do not offer open source solutions and have a somewhat complicated business structure for getting the products. They have a several implementations developed for specific solutions for filtering data in some relational databases, enforcing access control when to manage access to multiple applications and some others and a useful Eclipse IDE plug-in for easier creation of policies. The low point that was established while researching this company is that the products are not easy and straightforward to get to. On the other hand they often organize educational webinars for showing some uses and benefits of using ABAC. Their products unfortunately do not seem to be aimed at large scale IoT implementations as they do not mention those possibilities and the product aim at specific problems. On their site it is stated that they have a wide customer base, a number of offices across the US and have a goat to provide more ABAC solutions. Their products are focused on certain implementations mentioned before and unfortunately not a lot of information can be found on their official website [66], at least not regarding the core engine they are using for access control and technical details about their solutions. In any case this company offers great promise as it's focused is purely on dynamic access control utilizing ABAC and the OASIS XACML standard[3].

Balana

Balana is an XACML implementation. It is an engine that supports the latest version of the standard and is essentially it provides the functionality of the PDP and other core functionalities [67]. I is based on Sun's XACML implementation that will be addressed after Balana. Balana is a good engine that can also be expanded and it is used by lot of systems. It solves the most difficult and time consuming part of implementing a system utilizing XACML which is the PDP or more specifically the evaluation part. It an open source solution under the standard Apache licence. It was considered for using as a starting point instead of the AT&T project but wasn't because it doesn't support the JSON profile of the standard [22].

Sun's XACML Implementation

From what the research indicates this is one of the earlier implementations of XACML and is also an open source solution under a standard Apache licence. It was developed in Sun Microsystems Laboratories which is a part of Sun Microsystems, Inc. The official website [68] does not offer a lot of information but reveals that the last update was in 2006. This could mean many things but it is not wrong to conclude that further development will not occur. As Balana was based on this implementations and many other systems use Balana, its development will continue but probably inside other solutions that use one of these.

AT&T XACML implementation

This is an open source project available on GitHub [1], that implements XACML. This solution is not a complete implementation but offers a good base for further development and expansion. The documentation for this project is also lacking and is a big issue for anyone trying to use it but as it is the only solution that states support for the JSON variant of the XACML standard it was chosen as a base project for development. The design choices and reasoning is explained in more detail in subsection 4.1. This project has a partial implementation of all the features of XACML and has a large base of tests that prove the ones currently implemented work correctly. This means that certain more complicated conditions and combining algorithms will not be possible but it provides a good base for expansion.

This implementation consists of several projects covering different functionalities:

- XACML - contains base functionalities, base classes and interfaces. It contains the interfaces for PIP, PEP, PDP and PAP engines. Additionally it contains functionality for configuration and it is done trough a *xacml.properties* file located in the base folder of the

project. It is used by all of the other projects that have more specific functionalities implemented;

- XACML-PDP - contains all the functionality needed for the PDP to evaluate request. This includes calling PIPs and asking for requests
- XACML-PAP-ADMIN - contains functionality for the PAP;
- XACML-REST - contains common functionality used by the PAP-REST and PDP-REST projects. The functionality is regarding the REST service;
- XACML-PDP-REST - provides a REST service interface for the PDP;
- XACML-PAP-REST - provides a REST service interface for the PAP;
- XACML-TEST - contains a library of tests which verify the functionality of some components. The tests are mainly regarding the PDP and base XACML project and do not test the PAP or REST services.

As this project is a work in progress it doesn't include extensive documentation nor a straightforward method of running or using the functionalities that it provides. Many parts therefore are not finished and do not work. After considerable effort was taken to run and test all of the parts of this projects it was concluded that the PAP and the REST components either do not work or require significant effort and extensive knowledge of the project to utilize. The XACML and XACML-PDP and the functionality they provide on the other hand were tested and proven to work.

4 Solution description

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

In subsection 4.1 the solutions architecture and workflow will be explained along with design choices and technologies used, subsection 4.2 will give a description of every major component, subsection 4.3 will present the proposed implementation scenarios, subsection 4.4 will give a brief overview of development process and subsection 4.5 will present a critical overview and potential improvements.

4.1 *Design choices and general architecture*

After extensive research and regarding the technologies currently available, a number of design choices were made and a architecture was made. This subsection will describe the reasoning behind the decisions that were made.

The limitations that were 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). This is valid because it will still provide a proof of concept and in that case the limited functionality means limited evaluation possibilities like range searches. All of the major components are still present including all the communication and major functionality. 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.

4.1.1 Design choices and technologies used

This subsection will name the technologies used and present the reasoning behind choosing them.

Databases

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 2.5. 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.

XACML base project

The access control framework utilizes some functionality provided by the AT&T project [1] and uses some of its functionalities to make the developed solution work. A description of the AT&T project by components is given in subsection 3.2. The project was used mainly for its PDP engine, interfaces for the PIPs and configuration functionality. The PDP engine has the evaluation functionality built in along with asking PIPs for attributes while evaluating. The implementation required implementing that basic engine by extending the base class, initializing the PDP and making all the necessary configurations. Functionality of going through the policies and returning the result was built separately from the base project. The PIPs were built by implementing basic interfaces. Fetching of attributes and returning them to the PDP was built, the communication regarding which attributes are needed and which can be provided was also built. The configurations functionality was used to control the list of PIPs used, classes for the major components and other configurations.

4.1.2 Solution architecture

The architecture used is based on the one the one described in the standard's specification [3] and instead of having all of the functionality specified in the standard be implemented with a subset of simpler functionalities. Although the architecture is based on the architecture defined in the standard it was modified. The reasoning behind modifying the architecture will be explained in this subsection.

The architecture differs from the original one from the OASIS XACML standard in several areas. It doesn't contain the Obligation Service component, as can be seen in subsection 2.1.2. That component was dismissed as it is not essential for the basic functionality and it is not required for a

the targeted IoT implementations. It also contains the PRP (Policy Recovery Point) that is responsible for fetching policies and delivering policies. This is different from the standard's version as in that version the fetching of policies was done trough the PAP (Policy Administration point). In this architecture that functionality was removed from the PAP and moved to the PRP component. The reason for doing this were security reasons. If the PAP is left out of the evaluation process altogether and the PRP has only the functionality to read data, there is no chance of policy modification in that process. This is mainly done from a object oriented design perspective, and keeping within those principles. The principles in question are SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion) and the one that this decision is closest to is the *Single responsibility principle* and applying it to a component rather than just a single class. The PRP than has a single purpose and even for the engineer that is implementing or modifying this software, the chance of misuse is significantly less. As a result, the PAP is therefore an entity onto itself and is just an entry point for an administrator to manage policies. Also compliant with the *Single responsibility principle*.

In the next part an initial proposed architecture will be shown and described and after that the final architecture will be shown and described as well.

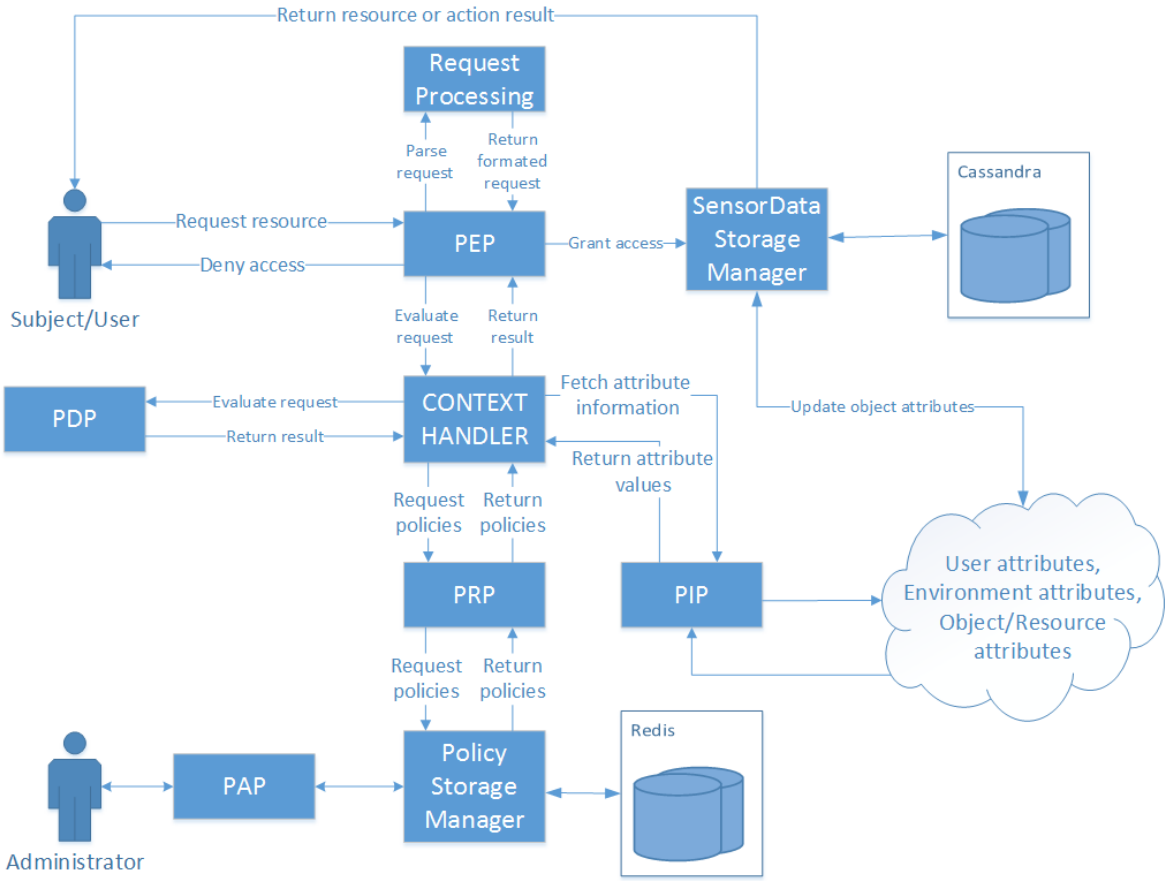

Figure 12. Architecture of the initial proposed solution

In Figure 12., a more detailed architecture of the initial proposed solution can be seen. This architecture is almost the same as the proposed one in the OASIS standard [3]. 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. Another issue is the connection between the context handler and the PRP. As a request is received the Context Handler has to get the policies and deliver them to the PDP. As it is just a "middle man" in this case, it is better for it to be connected directly to the PDP. Another issue is connection between the context handler and the PIP. The Context Handler could in this case could work in three ways:

1. Go through a list of policies, see if there are any attributes that are missing and fetch them from the PIP and deliver the modified request to the PDP for evaluation;
2. Fill the request "blindly" with every attribute that it can get with the attributes already contained in the request and deliver the modified request to the PDP for evaluation;
3. Wait for the PDP to ask for the attributes while it is evaluating, fetch the attributes from the PIP and deliver them to the PDP.

The 1st possibility is somewhat inefficient. The Context Handler has to have functionality for searching and parsing policies and request, while doing that it will go through all the policies and all fill the request with everything that it would need for all of the policies. The request could potentially become large and it would require significant time to parse trough everything and fetch, thus degrading performance. Also, this functionality is already built into the PDP as is does that while evaluating the requests.

The 2nd possibility is requires simpler functionality as it doesn't require to parse trough all of the requests but it fills every request with all possible attributes all the time. This will later be used by the PDP and it will slow down the evaluation process. Additionally, if there is a PIP that has to fetch attributes from another outside source trough a service and the process of fetching the

attributes takes significant time, that additional time will be added to the evaluation time. This will happen for every request evaluation therefore diminishing performance.

The 3rd way is equal to the one defined in the standard. This method is efficient as the PDP will ask for attributes only when it needs to and the Context Handler will deliver them to it. The problem in this scenario is that the Context Handler is still the "middle man" and just forwards requests and responses without having any additional functionality.

After some work was done, a final architecture was settled upon with slight modification to the initially proposed, and as a result, has slightly from to the architecture proposed in the XACML standard [3].

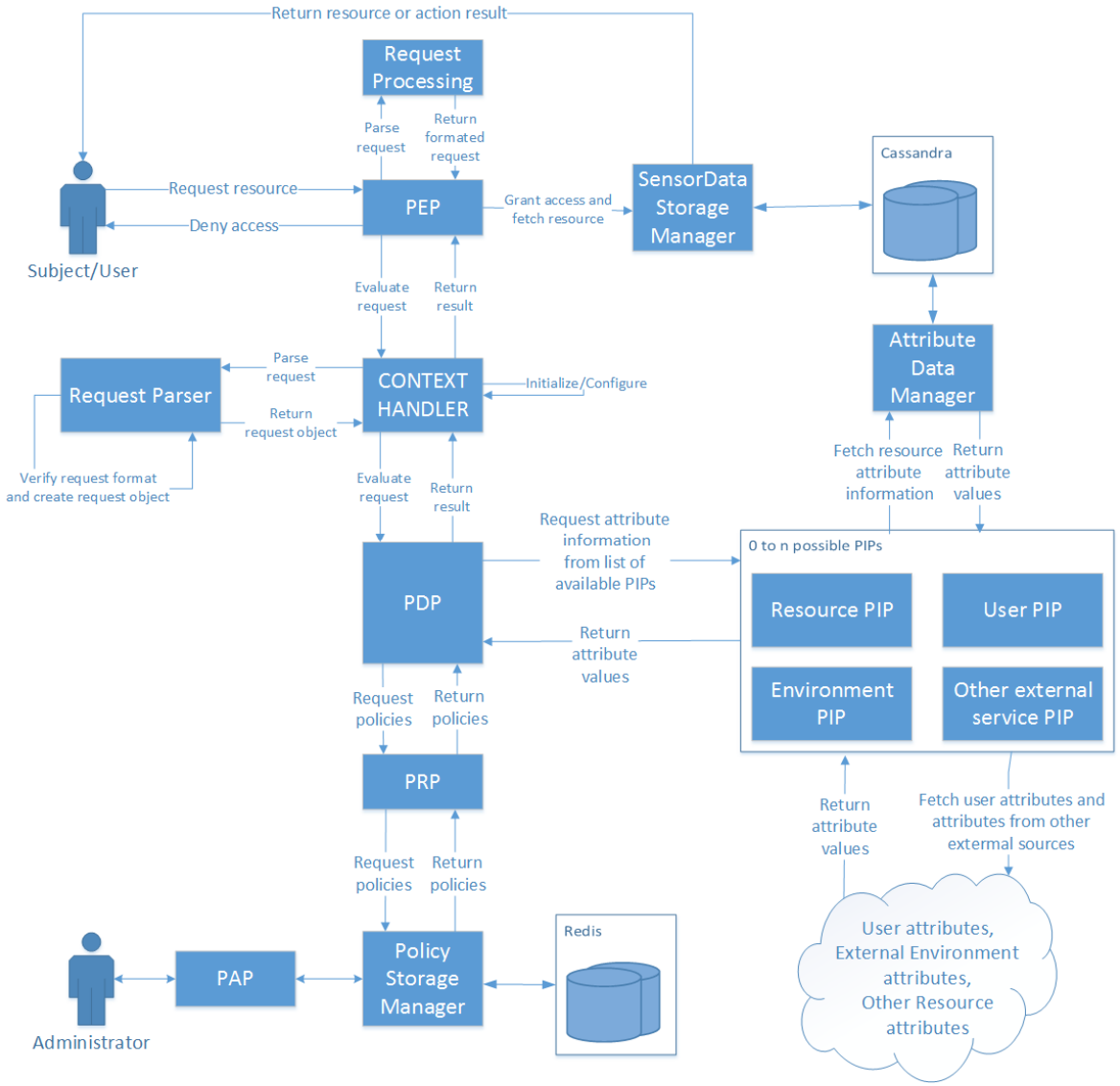

Figure 13. Architecture of the final solution

The final architecture can be seen in Figure 13. 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. 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. This architecture therefore deals with the issues identified in the initial one. The Context Handler maintains only a initialisation and configuration role rather that handling the workflow and being the "middle man". This was established as being more efficient and was adopted because of that. The PDP now fetches the policies and additional attributes directly from the PRP and List of PIPs, only when it needs to.

4.2 *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 13. that shows the architecture of the final solution.

4.2.1 Packages and dependencies

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

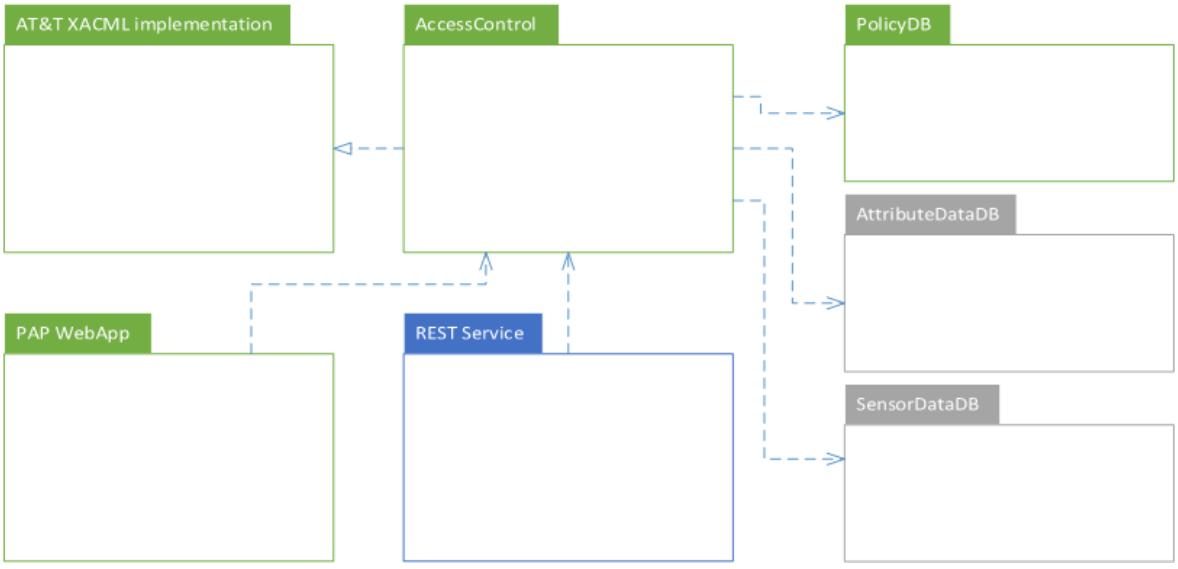

Figure 14. Project dependencies diagram between packages

In Figure 14. 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 because it is used, in one way or another, by all of the other ones.

The component listed are:

1. Helper - this component contains generic classes used by all other components. It contains: generic singleton and factory classes, functionality used for logging, basic and generic REST client implementation, abstract database manager class and factory, list of constants used, class containing useful list of functions like hashing, parsing and others;
2. AT&T - this is an open source project that was used for its functionality which can be described in detail in subsection 3.2. The main functionality used was the PDP engine, PIP interfaces so it can be used by the PDP engine and the configuration functionality. Details can be seen in subsection 4.1.1;
3. AccessControl - this contains the main functionality of this solution. It contains the PDP, PEPs, PIPs, PRP, PAP, Context Handler and others.
4. PolicyDB, SensorDB and AttributeDB - these contain the database managers and the java clients for the database they are using. The PolicyDB is in green as it is essential for the final solution, as policies need to be stored somewhere. The other two are not necessarily needed but are likely and meant to be utilized also. All three contain similar functionalities and in general are very similar;
5. REST Service - this component utilizes the Access Control to evaluate requests it receives. It is a simple REST service implementation. It is marked in blue because it's use depends on what type of integration is being utilized as described in subsection 4.3;
6. PAP Web App - this component serves the purpose of providing the administrator with an interface over which he/she can manage the policies used. It contains a web app with all of the standard classes and other files found.

This package layout was made so the functionalities can be utilized by other systems and were separate into different project simulating how they would be separated onto different machines.

4.2.2 Components

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 4.3 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.`

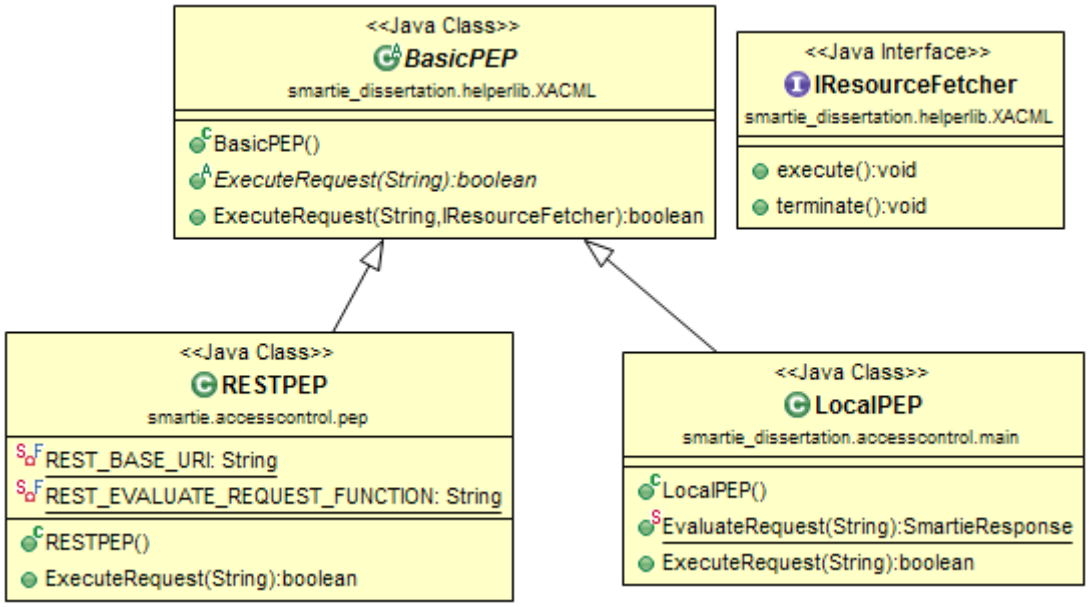

Figure 15. Class diagram of the PEPs

In Figure 15. 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 4.3.

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 subsection 4.1.

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.

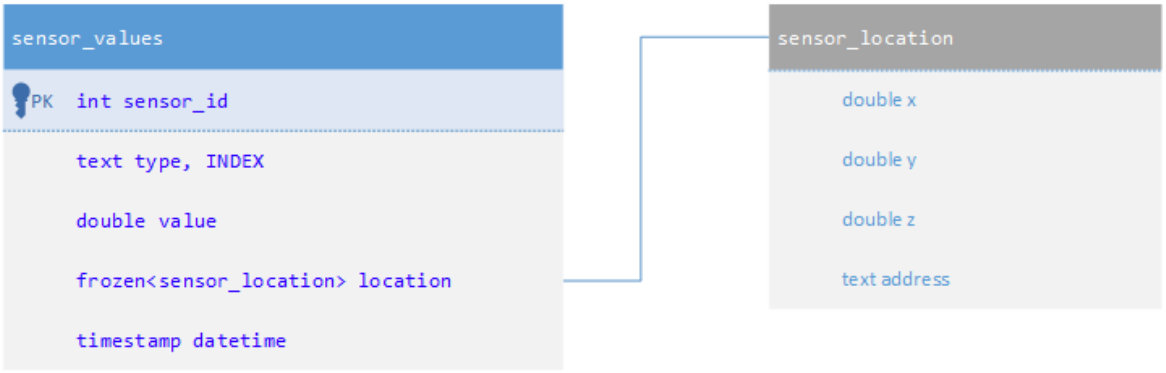

Figure 16. Cassandra database schema for storing sensor data

Figure 16. 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.

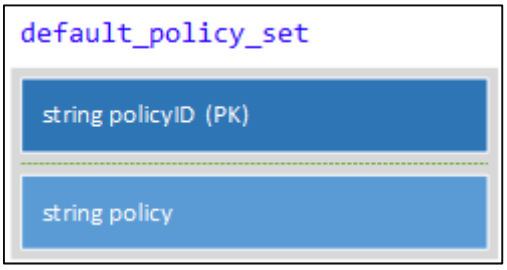

Figure 17. Redis schema for storing policies

Figure 17. 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.

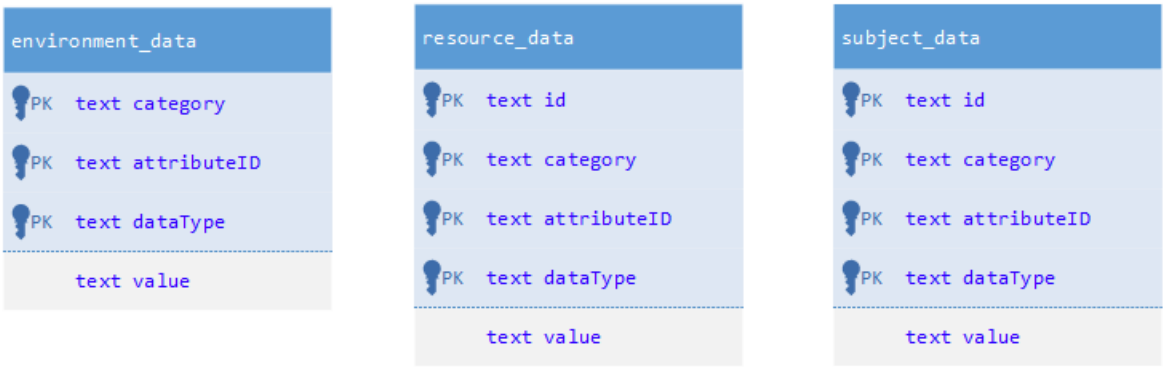

Figure 18. Cassandra database schema for storing attribute data

Cassandra is also used for storing attribute data as seen in Figure 18. 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.

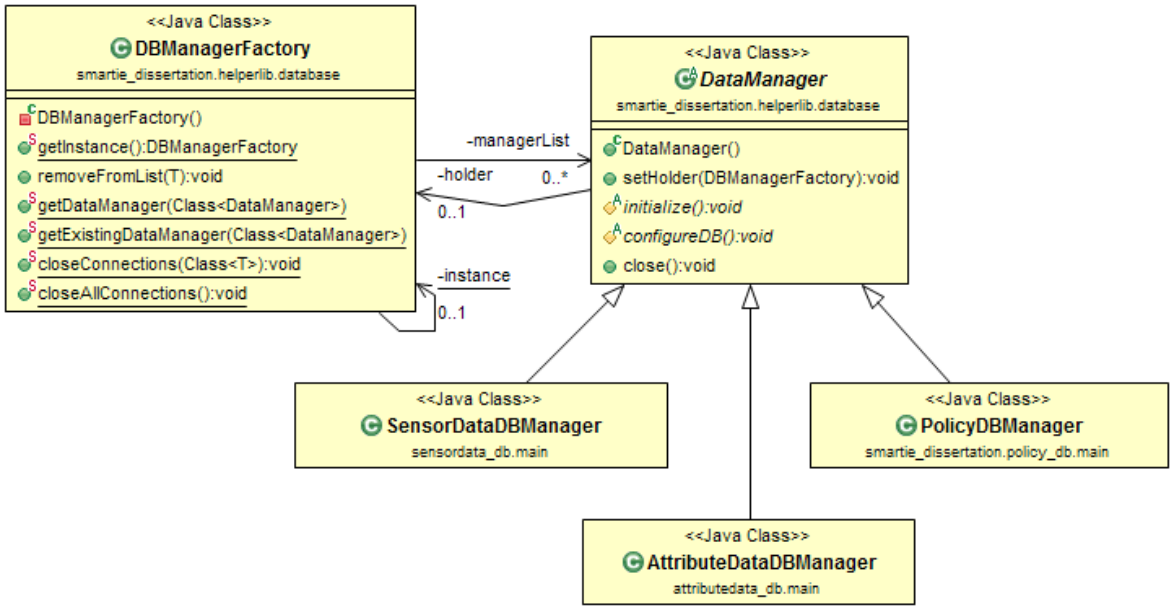

Figure 19. Class diagram for the Data Managers

The class diagram shown in Figure 19. 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. The PRP was realized in a simple manner but it leaves room for extending the functionality. For instance policy filtering according to the request's attributes.

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 [3]. 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 [1]. 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 [3] 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 [1] project and a implementation was built to suit the purposes of this solution. The building of the PDP required making configuration, connecting the PRP to get the policies and building the workflow used for evaluating against a set of policies.

The basic workflow of the PDP consists of the following:

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 20. PDP Workflow

As seen in the workflow in Figure 20. the PDP is set to allow the execution of the request if there is at least one policy that gives it permission. The workflow was built in this manner as it simplifies the process and policies that have to be made for a system.

The policies therefore have to be written with this workflow 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.

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.

4.3 *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. The scenarios that are shown are not the only possibilities but were determined to be the best ones. A discussion regarding scalability and possible distribution over more machines, along with other issues will be given later in this subsection.

4.3.1 Integrated solution

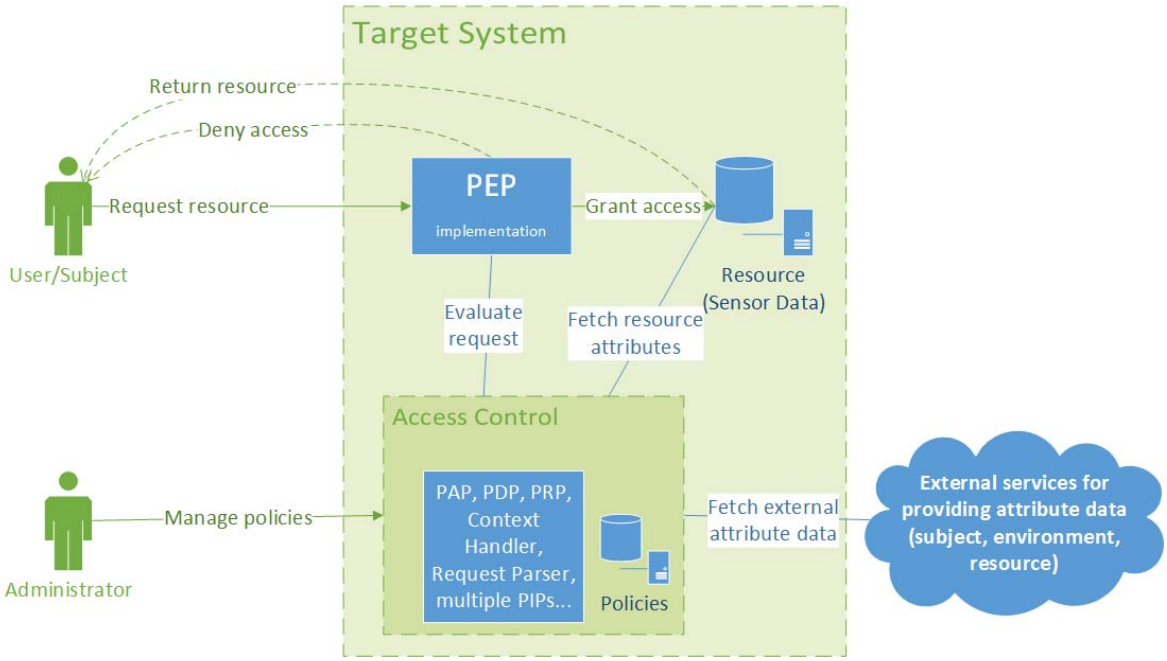

Figure 21. Integrated solution scenario schema

In Figure 21. 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 4.2. 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.

4.3.2 Using the solution as a service

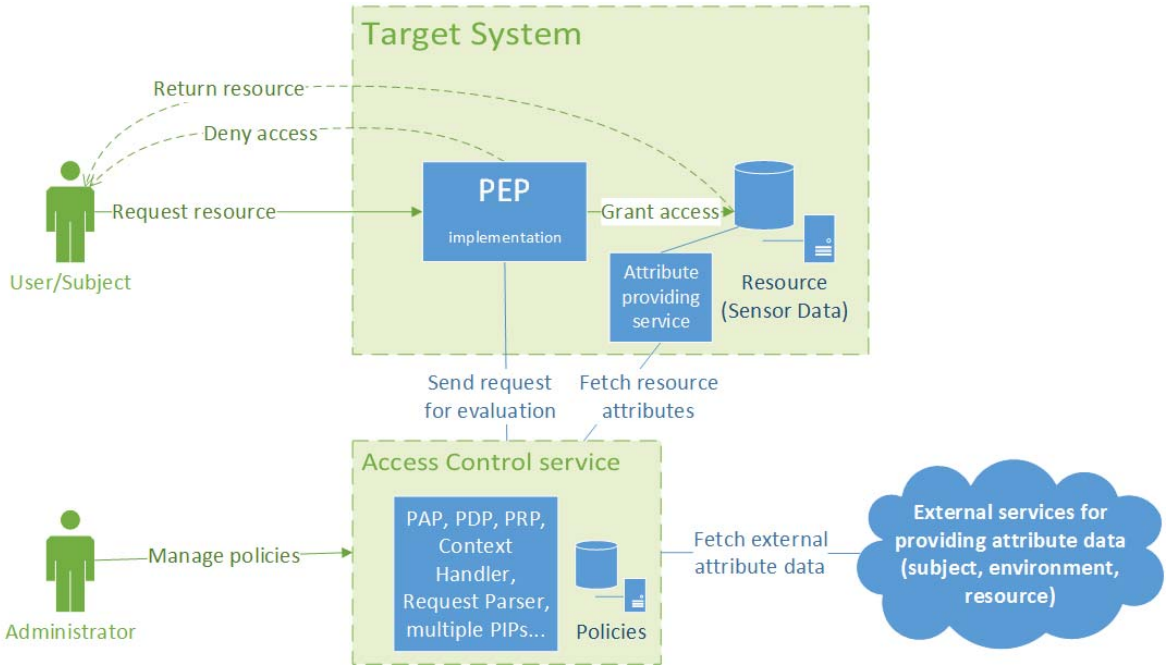

Figure 22. Schema of using the solution as a service

Figure 22. 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.

4.3.3 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.

4.3.4 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. 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.

4.3.5 Scalability and distribution

These scenarios were made and the components were grouped to run as a single entity for a reason. The components that should be grouped are: PDP, Context Handler, PRP and PIPs. These components are the essential components needed for evaluating the requests. Separation of these components would not bring any benefits but would bring only connection issues and possibly diminish performance. The PIPs can be connected to external services and fetch attributes from outside the system but should not be separated. Additionally the PAP, PAP web application, *Rest Service* and *PolicyDB* manager should be added to this group and should run on the same machine where the Redis database used for storing policies is running. Although this group isn't an essential part to the evaluation process they are endpoints that revolve around the database containing policies. Keeping these together with the rest of the group means keeping communication between components simple, fast and safe without the need of implementing additional safety measures. The PEP needs to be on the machine that is integrating access control.

This method of grouping these components brings up issues regarding scalability. Normally a distributed system scales much better that a non distributed system and if the components cannot be separated it is hard to have a distributed system. The solution to this would revolve around the replication capabilities of the Redis database used to store policies. The database can be replicated on multiple machines and multiple instances of the solution can run on all of those machines. This would then scale as needed. For this to work with the REST service an additional component would be needed. It would have functionality for handling the multiple instances and delegating the workload efficiently. This of course doesn't have to rely on the Redis database and can be exchanged for another storage solution if a better one is found. Because this can be viewed as a service for evaluating request against policies and is therefore a single "black box", it should stay that way and having multiple instances running on many machines has many benefits.

Along with scalability, the parallelisation of the process is an issue that has to be considered. This can be achieved using the same principle as before. Having multiple instances of a PDP and providing each one with a subset of policies and running everything parallel, is an easy and straightforward way to deal with the parallelisation issue. Long evaluation times in the case of a large set of policies can therefore be split in a fraction of the time by dividing the work and aggregating the result at the end.

4.4 *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, 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 [1] 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 5.

4.5 *Potential improvements*

At the end of Section 4.3 a number of issues were given regarding connections in the system. This developed solution in its current stage contains open and therefore unsecure connections for the REST services. These connections are a primary area of concern and have to be solved by implementing secure connections as explained in the same section 4.3.

Another area that would need improving is the list of possible logical operations available to the PDP. The current versions does not support some operations with certain data types. For example: more-equal and less-equal operations are not available for date/time data types. This list is not complicated to expand as it consists of extending the list of classes providing that functionality already available the same way as they were implemented.

The system also does not provide the possibility for multiple connections as it crashes in the case of multiple simultaneous requests. This could be solved by running multiple instances of

components and/or having a buffer for requests. Scalability and parallelisation of the evaluation process is discussed in subsection 4.3.5. Improvements would require running multiple instances of the system and multiple components of the PDP and having additional components handling and delegating the workflow efficiently. This would make the solution scalable and make the evaluation process run in parallel.

The last one is to provide the functionality for having multiple clients using the same service and distinguishing the policy lists used for everyone. This also applies for the PAP interface. This way of using the system could bring issues with the Redis database and it's memory limitations. This issue could be solved by hosting the service on machines with sufficient memory and/or having a different way of storing policies. The closest replacement in that case would be CouchDB. Performance issues with scaling should not occur as all of the internal components can run as multiple instances and replication characteristics of Redis are sufficient.

5 Proof of concept

This section will describe the test scenarios that were created to test the system and the test results will be presented. Along with that a description of a possible integration in SMARTIE will be given.

IoT applications are the systems that this solution was developed for. The solution can be integrated and used by other applications but IoT applications, and more specifically SMARTIE, will be considered as the main target system.

5.1 *Testing*

In this subsection the test Scenarios will be described and test result s will be shown and explained. The tests were designed to test the functionality and validate that the system is working as intended and also to test the performance characteristics. Also, both integration scenarios that are described in Section 4.3 were used to verify that they work and to make a performance comparison. The tests for the second integration scenario ("Integration as a service scenario") were done with by making calls from a SMARTIE component and using requests and policies already used by the system. This was done to verify that the solution and the integration scenario are working as intended when utilized by another system and also because SMARTIE was the primary targeted use case scenario. A brief description of the component is given in subsection 5.2.

2 types of tests were ran on every test scenario. The first was a qualitative test running a variety of requests. The goal of this test is verification of functionality or put more precisely, verifying that the system is behaving as expected and returning the expected responses. The requests vary in the complexity and also on the PIPs that their evaluation requires. Some do not require fetching additional attribute data others require attribute data from multiple sources so all aspects of the core of the framework and . The second test was a performance type of test and it consisted of repeating requests many times, some with a positive and some with a negative response. The response time was timed and because the responses were repeated, it was possible to extract a reliable result. It also has to be noted that the developed solution does not incorporate any type of caching so the repeating of the requests will not result in inaccurate results. It also has to be noted that performance wasn't a primary issue or concern while developing this solution, meaning that optimisations are more than likely possible. The request that were used for testing are named with the expected result added as a suffix. This means that a request with the suffix "*_permit*" or "*_all*ow" has the expected result *TRUE* and a request with the "*_deny*" has the expected result *FALSE*.

The last thing has to be mentioned is that these tests didn't include testing of the PAP Web Application/interface and PAP component.

The machine used for running local tests and for running/hosting the solution as a service is a HP proBook 4530s with a 8,00 GB of RAM, Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz 2.50 GHz processor package, running the Windows 7, 64bit operating system.

5.1.1 Local integration scenario testing

This scenario tested the local integration variant of the developed solution. From a performance perspective this is of course an ideal scenario so the results should reflect this.

#	Name	Expected result	Result
1	request1_Permit.json	TRUE	TRUE
2	request1_Deny.json	FALSE	FALSE
3	request2_Permit.json	TRUE	TRUE
4	request2_Deny.json	FALSE	FALSE
5	request3_Permit.json	TRUE	TRUE
6	request3_Deny.xml	FALSE	FALSE
7	request4_Deny.xml	FALSE	FALSE
8	request5_Deny..xml	FALSE	FALSE
9	request6_Allow.xml	TRUE	TRUE
10	request7_Allow.xml	TRUE	TRUE

Table 7. Qualitative results of the local integration scenario

#	Name	Expected result	Result	Response time (ms)
1	request_permit.xml	TRUE	TRUE	60
2	request_permit.xml	TRUE	TRUE	41
3	request_permit.xml	TRUE	TRUE	52
4	request_permit.xml	TRUE	TRUE	39
5	request_permit.xml	TRUE	TRUE	40
6	request_permit.xml	TRUE	TRUE	39
7	request_permit.xml	TRUE	TRUE	47
8	request_permit.xml	TRUE	TRUE	35
9	request_permit.xml	TRUE	TRUE	29
10	request_permit.xml	TRUE	TRUE	30
11	request_permit.xml	TRUE	TRUE	33
12	request_permit.xml	TRUE	TRUE	28
13	request_permit.xml	TRUE	TRUE	27
14	request_permit.xml	TRUE	TRUE	28
15	request_permit.xml	TRUE	TRUE	28
16	request_permit.xml	TRUE	TRUE	32
17	request_permit.xml	TRUE	TRUE	35
18	request_permit.xml	TRUE	TRUE	34
19	request_permit.xml	TRUE	TRUE	35
20	request_permit.xml	TRUE	TRUE	40
21	request_deny.xml	FALSE	FALSE	34
22	request_deny.xml	FALSE	FALSE	29
23	request_deny.xml	FALSE	FALSE	32
24	request_deny.xml	FALSE	FALSE	24
25	request_deny.xml	FALSE	FALSE	28
26	request_deny.xml	FALSE	FALSE	30
27	request_deny.xml	FALSE	FALSE	27
28	request_deny.xml	FALSE	FALSE	31
29	request_deny.xml	FALSE	FALSE	29
30	request_deny.xml	FALSE	FALSE	28
31	request_deny.xml	FALSE	FALSE	28
32	request_deny.xml	FALSE	FALSE	27
33	request_deny.xml	FALSE	FALSE	30
34	request_deny.xml	FALSE	FALSE	26
35	request_deny.xml	FALSE	FALSE	28
36	request_deny.xml	FALSE	FALSE	35
37	request_deny.xml	FALSE	FALSE	33
38	request_deny.xml	FALSE	FALSE	34
39	request_deny.xml	FALSE	FALSE	43
40	request_deny.xml	FALSE	FALSE	43
			Avrage:	**33,775**

Table 8. Performance test results of the local integration scenario

5.1.2 Integration as a service scenario testing

This scenario tested the "Using the solution as a service" integration scenario. The connection was done over a local network so the client and server machine had a direct connection. This scenario therefore doesn't take into account latency and other issues which would occur with a more realistic connection scenario.

#	Name	Expected result	Result
1	request1_Permit.xml	TRUE	TRUE
2	request1_Deny.xml	FALSE	FALSE
3	request2_Permit.xml	TRUE	TRUE
4	request2_Deny.xml	FALSE	FALSE
5	request3_Permit.xml	TRUE	TRUE
6	request3_Deny.xml	FALSE	FALSE
7	request4_Permit.xml	TRUE	TRUE
8	request4_Deny.xml	FALSE	FALSE
9	request5_Permit.xml	TRUE	TRUE
10	request5_Deny.xml	FALSE	FALSE
11	request6_Permit.xml	TRUE	TRUE
12	request6_Deny.xml	FALSE	FALSE
13	request7_Permit.xml	TRUE	TRUE
14	request7_Deny.xml	FALSE	FALSE
15	request8_Permit.xml	TRUE	TRUE
16	request8_Deny.xml	FALSE	FALSE
17	request9_Permit.xml	TRUE	TRUE
18	request9_Deny.xml	FALSE	FALSE
19	request10_Permit.xml	TRUE	TRUE
20	request10_Deny.xml	FALSE	FALSE

Table 9. Qualitative results of the "Using the solution as a service" integration scenario

#	Name	Expected result	Result	Response time (ms)
1	request_permit.xml	TRUE	TRUE	55
2	request_permit.xml	TRUE	TRUE	58
3	request_permit.xml	TRUE	TRUE	72
4	request_permit.xml	TRUE	TRUE	99
5	request_permit.xml	TRUE	TRUE	80
6	request_permit.xml	TRUE	TRUE	79
7	request_permit.xml	TRUE	TRUE	86
8	request_permit.xml	TRUE	TRUE	102
9	request_permit.xml	TRUE	TRUE	127
10	request_permit.xml	TRUE	TRUE	85
11	request_permit.xml	TRUE	TRUE	118
12	request_permit.xml	TRUE	TRUE	75
13	request_permit.xml	TRUE	TRUE	83
14	request_permit.xml	TRUE	TRUE	132
15	request_permit.xml	TRUE	TRUE	121
16	request_permit.xml	TRUE	TRUE	73
17	request_permit.xml	TRUE	TRUE	57
18	request_permit.xml	TRUE	TRUE	58
19	request_permit.xml	TRUE	TRUE	72
20	request_permit.xml	TRUE	TRUE	59
21	request_deny.xml	FALSE	FALSE	50
22	request_deny.xml	FALSE	FALSE	46
23	request_deny.xml	FALSE	FALSE	50
24	request_deny.xml	FALSE	FALSE	75
25	request_deny.xml	FALSE	FALSE	49
26	request_deny.xml	FALSE	FALSE	48
27	request_deny.xml	FALSE	FALSE	57
28	request_deny.xml	FALSE	FALSE	51
29	request_deny.xml	FALSE	FALSE	39
30	request_deny.xml	FALSE	FALSE	47
31	request_deny.xml	FALSE	FALSE	59
32	request_deny.xml	FALSE	FALSE	60
33	request_deny.xml	FALSE	FALSE	48
34	request_deny.xml	FALSE	FALSE	58
35	request_deny.xml	FALSE	FALSE	56
36	request_deny.xml	FALSE	FALSE	43
37	request_deny.xml	FALSE	FALSE	48
38	request_deny.xml	FALSE	FALSE	47
39	request_deny.xml	FALSE	FALSE	65
40	request_deny.xml	FALSE	FALSE	47
			Avrage:	**68,35**

Table 10. Performance test results "Using the solution as a service" integration scenario

5.1.3 Test results

The qualitative tests in both scenarios show that the system performs as predicted. These also validate that the fetching of additional data worked. It can be concluded that the PIPs and all of the other components are working as intended, of course, from a functional perspective.

The performance test show that the second scenario has slightly slower responses which is to be expected because of the overhead a REST service scenario brings. This difference in response time should increase in situations where the machines have more distance between, with high network traffic and other network related problems.

These tests showed that the developed solution performed as intended from a functionality perspective and satisfactory from a performance perspective, meaning that the overhead for the response times is acceptable for integrating in other systems. The tests that were done by making calls from the SMARTIE component were also a "proof of concept" test as the primary targeted system was SMARTIE. As the test show, the solution performed as predicted using requests and policies from the target system.

5.2 *Integration with SMARTIE*

This Section will give a general overview of SMARTIE and the *SmartData* platform from the information that was available. In Section 5.2.1 a brief description will be given of the state of the project during initial stages of doing this dissertation and Section 5.2.2 will give a critical overview and propose solutions that would improve upon the current implementation.

5.2.1 Current state and issues

SmartData is a platform that is being developed as a subsystem inside the SMARTIE project and it is responsible for storing sensor data and enforcing access control over the data and sensors. The system is being developed with scalability in mind and offers important functionality to SMARTIE. As the solutions for the problems this project faces are not readily available, and the final solution is meant to be used in systems used by many other systems and applications, using private and confidential data. Because of this, the system needs to provide a high level of security and availability.

In the current state, SmartData has access control for fetching data implemented by encrypting data and allowing decryption to users that have access to it and using oAuth for a combination of IBAC and RBAC. 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 one user or a set of users send requests for a large quantity of data, for which they don't have access to, all of that data needs to be fetched, policies need to be extracted and evaluation has to be done on every set of data. This is a big performance overhead as these operations require significant time so cumulatively it could be a opening for a DoS (Denial of Service) type of attack. This attack is just an example and is maybe not the only one that is possible.

Another issue is flexibility. In case of changing access policies for past data means fetching all the data for which the access policy needs to be changed and updating of the policies attached to it. In case of having a large policies, they contribute to a significant piece in the actual data being stored bringing unwanted overhead to the system. The current system doesn't predict that policies stored with data need updating and the system is still secure because the data will not be decrypted unless the subject wanting to access the data has access. This system in its current state therefore satisfies the current needs but isn't flexible enough to accommodate significant changes.

The publishing of data is done with a publish/subscribe type of workflow. Although this type of solution is appropriate for this type of scenario, it is somewhat limited on the type of conditions that can be integrated in the evaluation process. This comes back to the administration and system modifications usually required to accommodate complex conditions, decision making and also managing and updating of roles and other data depending on changes that take place with time.

Although the system is somewhat limited with flexibility and could have issues regarding overheads with administration, the planned product covers all of the requirements needed for it to function in the intended implementation.

5.2.2 Proposed implementation

As stated in Section 5.2.1 offers a good solution but has some potential problems, mainly regarding flexibility. This section will propose solutions for these issues by integrating the solution developed in this work.

A solution for the first issue mentioned (fetching encrypted data, "sticky policies") would be removing policies from the data and storing them in the policy database used by the solution developed in this work. The policies would need to be rewritten to have more descriptive and generic rules instead of regarding specific data. This would offer greater flexibility and because the policies could be managed on a single point, it would remove the need to always fetch data while evaluating policies. It would also mean that one policy could be responsible for more sets of data

reducing the number of evaluations required to access large sets of data. The last benefit would be reducing the amount of data needed to be stored because the policies would be stored on a different database and also because there wouldn't be the need of making many copies of the same types of policies but having one instead.

The performance tests shown in subsection 5.1.2 were done using policies and requests that are used by the current "sticky policies" solution and REST calls were made from that exact SMARTIE component. This means that the integration scenario by using the solution as a service was tested on this component. As the SMARTIE project isn't fully developed and not in production it was still not a "real world" test but it shows the integration scenario is and system worked as predicted.

The second proposal would be aimed to improve the current publish/subscribe solution for publishing of data. By integrating the solution developed for this work in the current one it would gain the flexibility for having more elaborate conditions and also reduce the amount of management needed making the system more secure in the long term. These integrations could be done with either integration scenarios described in Section 4.3 but because there is a possibility of sharing some policies, attribute sources and other functionality the second scenario would be easier to integrate and would offer an overall better solution.

6 Conclusion

This Section will give a brief overview and final remarks on the work that has been done and give thoughts and direction for future work.

6.1 *Final remarks*

Companies such as Google, Facebook, Amazon but also other organisation like in the EU in case of SMARTIE, are investing a lot of time, effort and money in solving the challenges regarding IoT, Big Data, Access Control and Security in all of these areas. These areas are currently on the forefront technological advancements and together with some other fields like AI (Artificial Intelligence) are considered areas that will bring the next push of advancements in technology in general. Security is an often overlooked area but still needs to be addressed in order to make the systems safe to use. Projects like SMARTIE offer great potential for applications that could significantly improve people's lives and security of personal information and critical systems is vital to making those real.

The solution that has been developed, although obviously not at production level is a proof of concept and demonstration of utilizing ABAC and the XACML standard in a IoT scenario environment, demonstrating a lot of benefits compared to other systems. Along with IoT scenarios, the applications that it could be utilized in are numerous. Just naming one example, with web applications and the OWASP (Open Web Application Security Project) top ten list of security issues, ABAC effectively solves half of them. Therefore, a standardised and simple to use solution could solve these issues with ease and minimal modification having significant impact of the level of security offered by basic services low budget services and applications.

The other benefit of utilizing ABAC and XACML is also longevity. Because of the flexibility offered by the diversity of policies that can be written and enforced, and the open nature of actual resource, security could be enforced over virtually anything from data, applications, services to sensors, actuators, locations... This open nature is probably the most significant contributor to the possible impact ABAC can offer. The solution developed in this work, as research indicates, unlike other solutions, implemented all of the major components defined in the OASIS XACML standard [3]. It was tested in various scenarios and offers 2 ways of integration, and both are straightforward and simple to do. Although this solution is a "proof of concept" and not a final piece of software, it doesn't mean that it is unusable as the tests demonstrate, but means that it can be used in the case of simpler scenarios and can be expanded for more demanding ones.

The ABAC methodology together with the XACML standard, has great potential and offers great benefits with virtually no downsides, which is not something that happens often. A finalized open source implementation that implements every aspect of the standard along with connectivity options with many types of services, would offer great benefits for many implementations, not only IoT applications as mentioned before. After building and having a secure system, verifying that it works correctly and predictably, the potential failure point is no longer directly a point in the system but the interfaces that system administrator and people implementing the solution have to use. The system's security relies primarily on correctly defined policies, making requests that correctly mirror the true requests and integration that is done correctly.

The human aspect of enforcing security is often overlooked as it is assumed that that aspect will work correctly. ABAC and XACML addressed this indirectly as the implementation should provide a solution that needs less maintenance together with policy and request formats that are readable and understandable for humans, which are significant improvements compared to current systems. The focus of future improvements therefore may shift from developing new systems from a system perspective but more from a usability, maintenance and human perspective in general which will be addressed in section 6.2.

6.2 *Future work*

The solution needs some improvements to make it usable for "real world" environments. Securing all of the connections for the REST services by utilizing TLS/SSL and/or oAuth or OpenID Connect as explained in the end of Section 4.3. would be the first step of first phase in improvements. Expanding the list of classes that provide the functionality for evaluation operations would be a secondary goal for improvements along with optimisations and adding the ability for handling multiple simultaneous calls in the service integration scenario.

After functionality and performance are expanded and improved, the focus of future work should be on the PAP. After the system is tested and it is confirmed that the system is working as intended, both from a functional and performance perspective, the only significant points of improvement are the PAP along with the interface for creation policies and a request parser component that would automatically mirror a variety of requests into JSON or XML formats of XACML requests. For example, the creation of requests from regular java code calls, url requests or others still have to be manually converted into standardised XACML requests. This requires a lot of effort, mainly because it requires the person implementing the solution to learn the standard

therefore making a possibility of incorrect request definitions more likely. This therefore makes the system dependant on the fact that a somewhat complex implementation is done correctly.

The improvement of the PAP interface has the same problems and as the whole security of a correctly implemented system relies on correctly defined set of policies, a complex interface indirectly will result in a less secure system. Axiomatics has some breakthrough in this area as its Eclipse plug-in ALFA provides a much simpler and easier way of writing XACML policies. The next step beyond their plug-in would be a completely graphical interface that would be aimed more towards people that do not necessarily need to have an in depth knowledge of XACML or ABAC. As a satisfactory system is created and becomes available for easy and wide spread implementation the focus of future developments may shift to the human aspect of maintaining and integrating security systems because even a perfect security system cannot work if it is used incorrectly. Because of this future systems could move towards removing the need for administration altogether and provide security to resources in a totally automated way.

7 References

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

[2] R. S. a. R. K. Xin Jin, "RABAC : Role-Centric Attribute-Based Access Control," *Lecture Notes in Computer Science Volume 7531*, pp. 84-96, 2012.

[3] "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].

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

[5] "Redis official website," Redis, [Online]. Available: http://redis.io/. [Accessed 20 February 2015].

[6] "MySQL homepage," Oracle Corporation, 2015. [Online]. Available: https://www.mysql.com/. [Accessed March 2015].

[7] "XAMPP," Apache Friends, 2015. [Online]. Available: https://www.apachefriends.org/index.html. [Accessed March 2015].

[8] "PostgreSQL," The PostgreSQL Global Development Group, 2015. [Online]. Available: http://www.postgresql.org/. [Accessed March 2015].

[9] "Axiomatics ALFA home," Axiomatics, 2015. [Online]. Available: http://www.axiomatics.com/solutions/products/authorization-for-applications/developer-tools-and-apis/192-axiomatics-language-for-authorization-alfa.html. [Accessed Fevruary 2015].

[10] "Jersey homepage," Oracle Corporation, 2015. [Online]. Available: https://jersey.java.net/. [Accessed March 2015].

[11] "JAVAX-RS," Oracle Corporation, 2014. [Online]. Available: https://jax-rs-spec.java.net/. [Accessed February 2015].

[12] "Maven homepage," The Apache Software Foundation, 2015. [Online]. Available: https://maven.apache.org/. [Accessed March 2015].

[13] R. K. R. S. Xin Jin, “A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC,” *Data and Applications Security and Privacy XXVI*, pp. 41-55, 2012.

[14] S. Osborn, “Mandatory Access Control and Role-Based Access Control Revisited,” *Proceeding RBAC '97 Proceedings of the second ACM workshop on Role-based access control*, pp. 31-40 , 1997.

[15] J. S. R. K. Rui TU, “An Identifier-Based Network Access Control Mechanism Based on Locator/Identifier Split,” *Int. J. Communications, Network and System Sciences*, pp. 641-644, 12 July 2009.

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

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

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

[19] S. K. L. A. M. M. C. T. Kathi Fisler, “Verification and Change-Impact Analysis,” *ICSE '05 Proceedings of the 27th international conference on Software engineering*, pp. 196-205, May 2005,.

[20] S. P. R. L. D. K. S. S. Markus Lorch, “First Experiences Using XACML for Access Control in Distributed Systems,” *Proceeding XMLSEC '03 Proceedings of the 2003 ACM workshop on XML security*, pp. 25-37, 2003.

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

[22] “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].

[23] N. W. Lu Tan, “Future Internet: The Internet of Things,” in *3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, 2010.

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

[25] 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].

[26] 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).

[27] 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.

[28] 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.

[29] 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].

[30] A. G. S. L. A. B. Michele Zorzi, "FROM TODAY'S INTRANET OF THINGS TO A FUTURE INTERNET OF THINGS:A WIRELESS- AND MOBILITY-RELATED VIEW," *IEEE Wireless Communications*, pp. 1536-1284, December 2010.

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

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

[38] G. Lafuente, "The big data security challenge," *Network Security*, January 2015.

[39] "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.

[40] 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.

[41] H. K. S. R. R. C. Konstantin Shvachko, "The Hadoop Distributed File System," *Mass Storage Systems and Technologies (MSST)*, pp. 1 - 10, 3 May 2010.

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

[43] "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].

[44] J. D. a. S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, pp. 107-113 , January 2008.

[45] "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].

[46] S. B. A. D. Sandhya Narayan, "Hadoop Acceleration in an OpenFlow-based cluster," in *SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012.

[47] "MongoDB HomePage," MongoDB, Inc., [Online]. Available: http://www.mongodb.org/. [Accessed 7 February 2015].

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

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

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

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

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

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

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

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

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

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

[58] 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.

[59] 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.

[60] "A comparison between several NoSQL databases with comments and notes".

[61] 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.

[62] D. CORPORATION, "Benchmarking Top NoSQL Databases, A Performance Comparison for

Architects and IT Managers," 2013.

[63] T. N. &. T. A. Pardo, "Conceptualizing Smart City with Dimensions of Technology, People, and Institutions," *The Proceedings of the 12th Annual International Conference on Digital Government Research,* pp. 282-291, June 2011.

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

[65] A. Z. L. V. M. Z. Angelo Cenedese, "Padova Smart City: An urban Internet of Things experimentation," *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium,* pp. 1 - 6, 19 June 2014.

[66] "Axiomatics Hamepage," Axiomatics, 2015. [Online]. Available: http://www.axiomatics.com/. [Accessed 20 May 2015].

[67] "WSO2 Balana Implementation GitHub project," [Online]. Available: https://github.com/wso2/balana. [Accessed 20 May 2015].

[68] "Sun's XACML Implementation," Sun Microsystems, Inc, 16 July 2006. [Online]. Available: http://sunxacml.sourceforge.net/. [Accessed 20 May 2015].

[69] 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..

[70] "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].

[71] 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 .

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

[73] 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.

[74] 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.

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

[76] "XML website," O'Reilly Media, Inc., [Online]. Available: http://www.xml.com/. [Accessed 20 Merch 2015].

[77] "xacml project," [Online]. Available: http://xacmlinfo.org/2014/11/30/implementing-rbac-and-abac-with-xacml/. [Accessed 20 May 2015].

[78] H. Lindqvist, Mandatory Access Control, Ume˚a University, Department of Computing Science, 2006.

[79] A. a. B. N. a. C. A. P. a. V. L. a. Z. M. Zanella, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, 2014.