Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2014

Bruno Miguel Cunha Faria

Plataforma de Gestão M2M

M2M Management Platform

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

**Bruno Miguel
Cunha Faria**

Plataforma de Gestão M2M

M2M Management Platform

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor João Paulo Silva Barraca, Professor assistente convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Aos meus pais, porque sem eles nada seria possível...

o júri / the jury

presidente / president	Prof. Doutora Susana Isabel Barreto de Miranda Sargento professora associada da Universidade do Aveiro
vogais / examiners committee	Mestre Ricardo Azevedo Guerra Raposo Pereira especialista na Pt Inovação e Sistemas
	Prof. Doutor Diogo Nuno Pereira Gomes professor auxiliar da Universidade do Aveiro (Co-Orientador)

agradecimentos / acknowledgements

Gostaria de começar por agradecer aos meus orientadores, Prof. Diogo Gomes e Prof. João Paulo Barraca, pelo acompanhamento prestado ao longo deste trabalho, e por me terem dado a possibilidade de crescer a nível profissional. Agradeço também ao grupo ATNoG e a todos os seus elementos e amigos pelo bom ambiente que proporcionaram neste ano de trabalho. Queria também agradecer ao meu Pai, à minha Mãe e Irmão, pelo apoio incondicional. E por fim, obrigado Helena, por todos estes anos ao meu lado e por me ajudares a terminar esta jornada. A todos, mais uma vez, obrigado.

Palavras Chave

Máquina-a-Máquina, Internet das Coisas, Plataformas intermédias de integração, Aquitectura Orientada ao Serviço.

Resumo

A Internet das Coisas continua a ser uma área em grande crescimento e de grande interesse. Estão constantemente a surgir novas soluções e inplementações, tanto ao nível dos serviços como ao nível das comunicações Máquina-a-Máquina, promovendo assim o aparecimento de novos modelos de negócio. Desta forma surgiu naturalmente a possibilidade de abstrair a gestão de sensores da criação de serviços. Permitindo assim, uma delagação da gestão por parte de empresas detentoras de sensores, para se focarem no conteúdo com a criação de serviços. Contudo esta divisão acarreta algumas preocupações de segurança quanto ao controlo de acesso. Nesse sentido, esta dissertação propõe uma possível solução para o mesmo, englobada numa plataforma orientada ao serviços interligada com uma solução ETSI M2M. Promovendo a interoperabilidade entre sensores e permitindo assim uma grande elasticidade na criação de serviços.

Keywords

Machine-to-Machine, Internet of Things, Middleware integration platforms , Service-oriented architecture.

Abstract

The Internet of Things is still a fast growing area and topic of interest. New solutions and implementations keep emerging, both in service oriented solutions or device oriented solutions with M2M communications, therefore promoting the creation of new business models. Thus, as a natural evolution, came the possibility to abstract sensor management from service creation. Allowing a delegation of sensor management from the sensor providers, to focus on content creation through services. However, this delegation brings new concerns regarding access control. Consequently, this dissertation proposes a possible solution to this problem, enclosed in a service oriented platform interconnected with an ETSI M2M solution. Promoting interoperability between sensors and allowing a great elasticity in service creation.

Contents

List of Figures

List of Tables

Acronyms

ACL	Access Control List
BPMN	Business Process Model and Notation
BPMS	Business Process Management Systems
CDI	Contexts and Dependency Injection
CoAP	Constrained Application Protocol
DA	Device Application
DARPA	Defense Advanced Research Projects Agency
DSCL	Device Service Capability Layers
DSN	Distributed Sensor Networks
EAI	Enterprise Application Integration
EIP	Enterprise Integration Patterns
EIS	Enterprise Information Systems
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standard Institute
GA	Gateways Application
GSCL	Gateway Service Capability Layers
IAM	Identity and Access Management
IdM	Identity management
IdP	Identity Provider
IoT	Internet of Things
IP	Internet Protocol
JAX-RS	Java API for RESTful Web Services
JCA	Java EE Connector Architecture
JMS	Java Message Service
JPA	Java Persistence API
JSF	JavaServer Faces
JSP	JavaServer Pages
LDAP	Lightweight Directory Access Protocol
M2M	Machine to Machine
MOM	Message-Oriented Middleware
NA	Network Application
NSCL	Network Service Capability Layers
OASIS	Advancing open standards for the information society
PaaS	Platform as a Service
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
QoS	Quality of Service

RBAC	Role-based Access Control
ROI	Return On Investment
SAML	Security Assertion Markup Language
SCA	Service Component Architecture
SCDL	Service Component Definition Language
SCL	Service Capability Layers
SDO	Standards Developing Organization
SOA	Service-oriented Architecture
SP	Service Provider
SQL	Structured Query Language
SSO	Single Sign-on
STS	Security Token Service
TC	Technical Committees
WS-BPEL	Web Services Business Process Execution Language
WSN	Wireless Sensor Network
XACML	eXtensible Access Control Markup Language

CHAPTER 1

Introduction

The evolution and growth of communications, has propelled a large number of small devices to be increasingly more connected. Devices like smartphones, tablets, televisions and even fridges, can now be connected to the Internet and therefore exchanging information with the rest of the world. This evolution is reaching small scale devices like sensors and actuators, making them connected devices, turning them into smart things.

This so called smart things or smart objects, have been helping the growth of the Internet of Things (IoT) concept, providing a digital sense of the physical world. They can be termed as real-world services, for their capability to provide near real-time state of the physical world. However, despite the quantity and quality of the information produced by those "things", without the proper context, this information has no meaning.

To tackle the problem of interconnecting (Machine to Machine (M2M) communications) and interpret the data coming from this so called smart things, platforms that allow interoperability and cataloging of information started to emerge. Moreover, enabling applications and services to use and operate over the information generated by those devices.

1.1 MOTIVATION

The ever increasing number of connected devices, brings endless new possibilities to old problems, ranging from industry optimizations to humanitarian issues like health and quality of life. Although, for this to happen, interpretation of this information is needed as well as compilation of multiple information to obtain further results.

To promote even more new connected devices and to bridge the gap between raw data and enriched data, a new concept of solution has been emerging, the IoT Platform, where the device management is binded with an application platform, potentiating the creation and deployment of enriched applications and services. Therefore, as consequence, promote fine grained access control, in order to secure these highly scalable platforms.

1.2 OBJECTIVES

The focus of this dissertation is centered in the development of a Service Platform integrated with a M2M middleware to enable the aforementioned bridge between devices and services. Furthermore, this dissertation also aims to provide a solution for access control as well as service mediation and data integration.

Additionally, this document provides an objective analyses on some of the most relevant solutions in this area.

1.3 DISSERTATION STRUCTURE

This dissertation is composed by 6 chapters, being the first chapter the already presented introduction, the remaining chapters goes as follows:

- **Chapter 2:** presents a description about the state of the art, the evolution of M2M and the current state of IoT, existing technologies in the Identity and Access Management (IAM) area as well as the current trends Service-oriented Architecture (SOA);
- **Chapter 3:** provides a complete overview about the implemented architecture, describing in detail each component and how everything fits together;
- **Chapter 4:** gives a detailed explanation about the implementation, the technologies used and the workflows it supports;
- **Chapter 5:** describes a test scenario, the stress tests performed, the results obtained and an analyses about those same results;
- **Chapter 6:** supplies the final conclusions about the developed work.

CHAPTER 2

State of the Art

2.1 MACHINE TO MACHINE

M2M refers to technologies that allow communication between devices of the same type, via wired or wireless communication networks, all without human intervention. It is not a new concept, since it is already common to have communications between machines like routers and servers, or telemetry systems. Although today, the concept for M2M goes a little further, not only machines can talk with each other, they are also able to give meaning to information they receive. This opened a new world of opportunities, in which the machines will help us humans, to better understand the world we live in.

The evolution of the M2M term, comes as a result of the technological progress over the last decades, including the decreasing costs of semiconductor components, the uptake of the Internet Protocol (IP) and the dissemination of the Internet to all types of devices[1].

The decreasing costs of semiconductor components as well as the miniaturization of electronic circuits, led to a proliferation of small scale electronic devices. Sensors and actuators, that were before big and some how expensive, can now be tiny to the level of milli or even nano scale[2] as well as low on cost. This enabled the deployment of thousands of sensors and actuators in multiple scenarios, creating an opportunity to start sensing the physical world, as never before.

Moreover, the increasing adoption of IP as well as new transport protocols like Constrained Application Protocol (CoAP) oriented for less capable devices, as thrived the creation of larger sensor networks in wired and wireless environments. These networks, allow a better understanding of the surroundings, and enables the integration of these devices with household appliances, such as fridges and coffee machines, making them smarter and capable of making decisions, or even notify the house owner, e.g., on lack of food supplies. Environmental, health and industry are also great candidates for these types of networks.

In the last decades, the advances in cellular networks, created a new paradigm of always connected devices, being these devices smartphones, tablets and smartwatches, these are the so called smart devices. The evolution in M2M, are turning ever more, simple devices like sensors and actuators into smart devices, and therefore connecting them to the Internet. As such, by connecting these "things" to the Internet, thrives the so called IoT.

2.2 WIRELESS SENSOR NETWORKS

A Wireless Sensor Network (WSN) is a network consisting of numerous sensor nodes, scattered in an unattended environment (i.e. sensing field), capable of sensing the physical world and communicate that data wirelessly to an infrastructured network[3]. A sensor node can also aggregate actuators to be able to interact with the physical world.

The term WSN is not new, its origin can be traced back to the Distributed Sensor Networks (DSN) program at the Defense Advanced Research Projects Agency (DARPA) at around 1980. DSNs were composed by many spatially distributed nodes, being each node a low-cost sensing node operated autonomously but in collaboration with other nodes[4]. At that time, it was a very ambitious program given the state of the art, as it was before personal computers and workstations, the processing was done mostly on minicomputers and the Ethernet was just becoming popular[4].

When environment variables can be monitored - temperature, humidity, pressure, etc - a use case for WSN can be applied, such as animal tracking, forest surveillance, flood detection and weather forecasting[5]. As an example, to better understand climatic change, involving sea-level alteration as a consequence to global warming, researchers from University of Southampton, have built a glacial environment monitoring system[6], using sensors within the ice and the sub-glacial sediment, creating a WSN and avoiding the use of wires to reduce any possible disturbance to the environment.

2.3 INTERNET OF THINGS

The Internet of Things concept, despite not having an universal definition[7], in its core, it represents the everyday objects with capacities to sense, process and transmit information between other devices and through the Internet, all without human mediation. The closeness of this definition with the one from M2M (section 2.1), depicts the why of the inner relationship between the terms and the confusion it sometimes generates. As it is the M2M technology that makes IoT possible, and it is the interest around the IoT that pushes the M2M evolution and adoption even further.

Recent studies by GSMA, states that M2M accounts for one in ten mobile connections in the US and one in 20 in Oceania and Europe [8]. Furthermore, a 2013 study from Machina Research, estimates that the global M2M revenue in 2022 will reach USD1.2 trillion, up from USD200 billion in 2013[9]. This represents a huge business opportunity and reflects the investment and expansion in M2M, as well as the projections for its exponential growth[9]. As an example of the increasing interest in this area, in Sweden, one in four mobile connections are from M2M devices[10], this is a result of legislation, as it was ruled that each household should be able to accurate monitor monthly electricity using smart meters.

As another consequence to the interest around IoT, is the extensive research thats being done on IoT and M2M. A recent study tried to understand the trends and paths by categorizing research topics on IoT [7]. Figure 2.1 depicts a pie chart with the referred categorization. As stated in the pie chart, technology appears as the top researched area, and despite being a wide topic, the focus is mainly on system architecture and interoperability. This indicates the importance of interoperability and the work being done to tackle that problem.

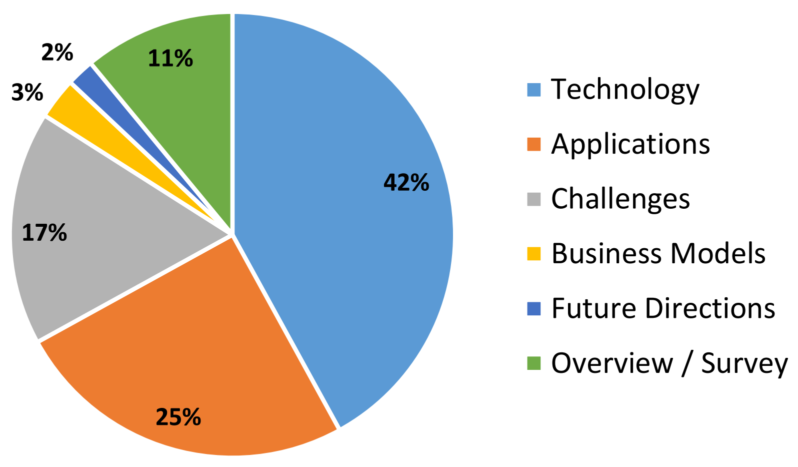


Figure 2.1: Distribution of articles by major category [7]

2.4 FROM M2M TO IOT

With the increasing interest around IoT and M2M, comes the necessity for greater sensor networks, capable of providing data in a smartly manner to enable content creation and applications development.

As a first approach and with the already existing WSN (section 2.2), was relatively easy and direct to create a system capable of generating data (sensors) and use that data (applications or services). Having the WSN as base, can provide network functionalities to applications, creating an highly coupled system where applications communicate directly with the sensors in the WSN. This system model is referred to as vertical (represented in

Figure 2.2), its initial simplicity lead many to its adoption. However, this model is not capable of integrating with other solutions and it is not easily scalable, since the lack of abstraction, the applications are binded to the sensors and network specifications.

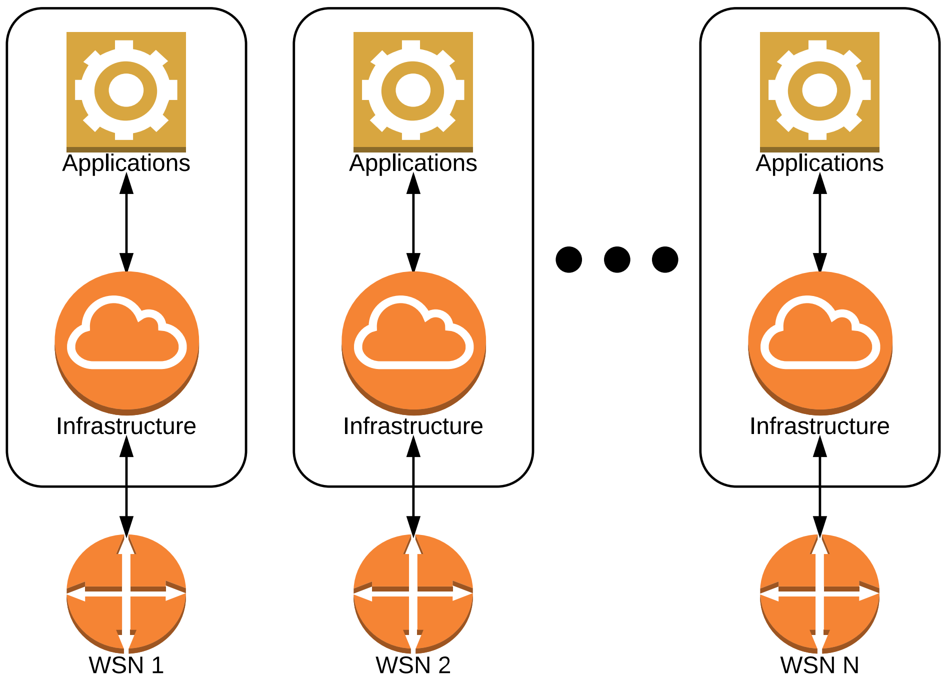


Figure 2.2: Example of a vertical solution

As an evolution to the limited vertical solutions, a new model of solution started to arise - the horizontal model. This solution, creates a layer of abstraction between applications and sensor networks, therefore decoupling the applications from the network specifications (Figure 2.3). This originates a more flexible system, capable of scaling, easing management and improved maintainability.

The horizontal approach is a great improvement from the previous vertical solutions and today it is the best choice for creating IoT Platforms. However, the lack of specifications or standard, to define the platform structure and interfaces, as well as to recommend the use of open protocols, can jeopardize interoperability between solutions, devices and applications. The following section will present the efforts being done to standardize M2M and therefore IoT.

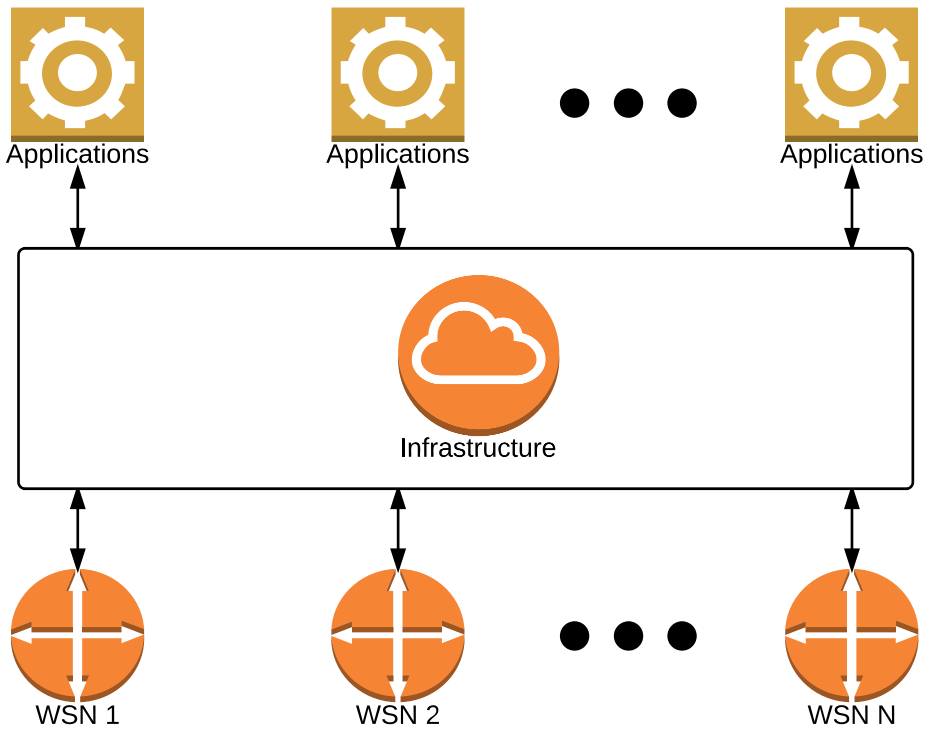


Figure 2.3: Example of an horizontal solution

2.5 EUROPEAN TELECOMMUNICATIONS STANDARD INSTITUTE

ETSI is a not-for-profit organization, funded in 1998 and recognized by the European Union as a European Standards Organization. It produces globally-applicable standards for Information and Communications Technologies.

The organization handles standardization in different areas of expertise by creating different Technical Committeess (TCs). Each of those committees is responsible for conducting work in that specific area.

In 2008 after detecting the interest M2M was acquiring and it's possible exponential growth, the TC for M2M was created, assuming then the importance of M2M and making it a topic of interest. ETSI was the first Standards Developing Organization (SDO) to address M2M, and other organizations only started tackling M2M standardization in subsequent years.

The creation of standards aims to improve interoperability, but multiple standards can have the opposite effect. To avoid this problem, a global initiative by the name of oneM2M was created and signed on July 2012 by seven major SDOs (ETSI included). This initiative has the objective to join forces in creating a single unified standard. To finalize, oneM2M delineated an extensive list of goals and benefits from a cooperated M2M standard [11], for brevity, the following list represents a summarized list of those topics:

- Develop and consolidate a common M2M Service Layer by creating Technical Specifications and Technical Reports.
- Collaborate with wireless and wireline SDOs and fora responsible for developing standards for core and access networks.
- Boost economics of scale in the context of M2M.
- Simplify application development.
- Reduce market fragmentation, enhance interoperability as well as security.

2.5.1 M2M STANDARD

The M2M Standard has the objective of creating a common ground for scalable and interoperable M2M systems, providing a normalized high-level architecture as well as interfaces and protocols that prove to be adequate for these types of systems. According to ETSI, the substitution of proprietary vertical architectures by standardized horizontal architectures is necessary to achieve interoperability. Moreover, giving M2M applications a common infrastructure, environments and network elements. Improving maintenance of devices connections, M2M applications as well as scalability. In terms of security, do to its importance, an approach from the beginning was necessary in order to prevent future incongruities of the standard, that would culminate in incompatibilities. ETSI M2M standard defines two types of security, one to handle Service Bootstrap and Connection procedures, while the other is responsible to ensure authorized access to data using a permission system. Using both mechanisms, secure communications between authenticated endpoints and authorized applications can be attained.

2.5.2 HIGH-LEVEL ARCHITECTURE

The M2M standard divides the high-level architecture in two main domains - the network domain and the M2M devices/gateway domain. For ETSI a layer that provides M2M functionalities is a Service Capability Layers (SCL), therefore the entity responsible for the network domain is the Network Service Capability Layers (NSCL) and for the other domain we have Device Service Capability Layers (DSCL) and Gateway Service Capability Layers (GSCL)

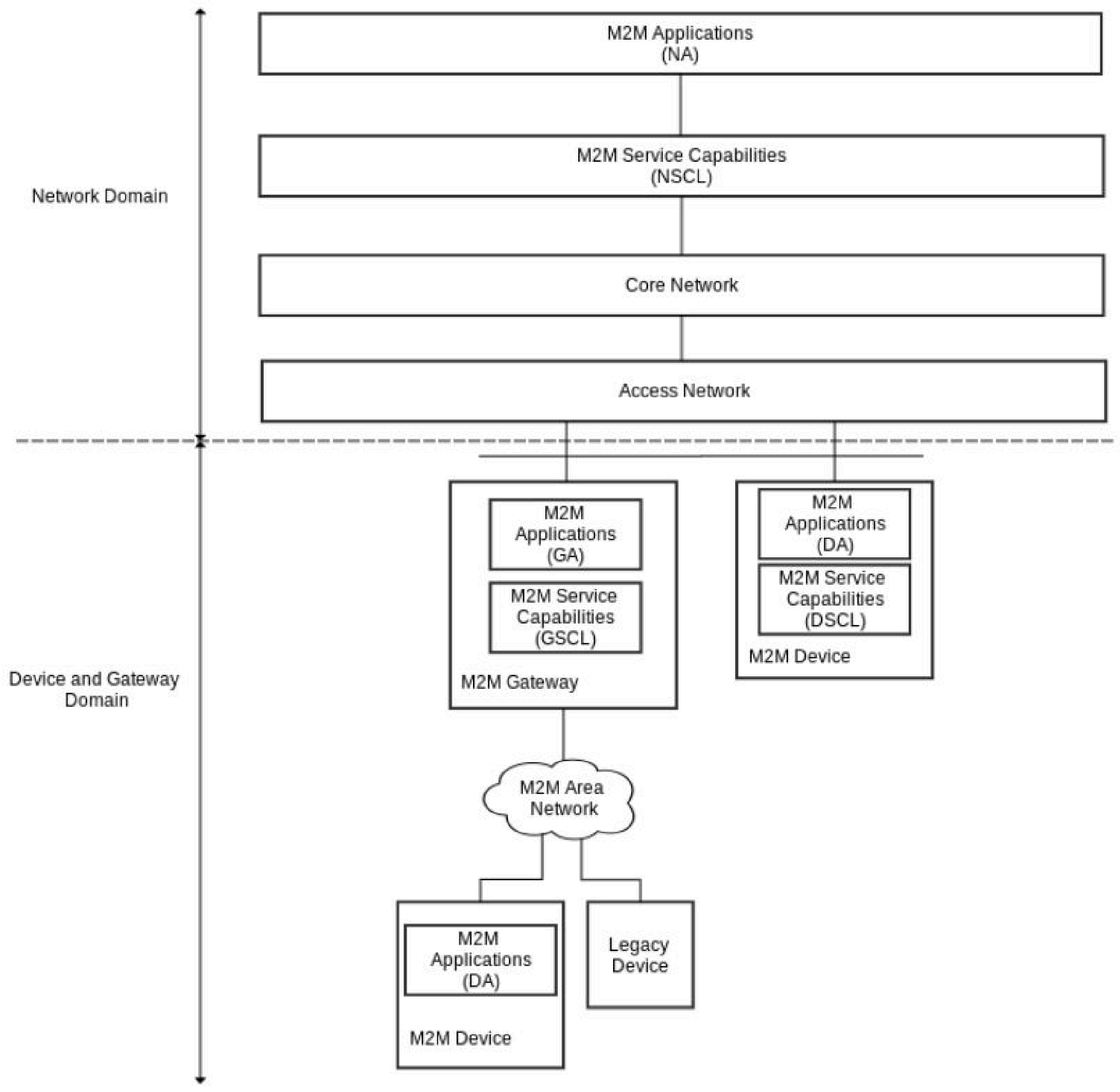


Figure 2.4: High level architecture for M2M [12]

accordingly. The two main domains as well as their division are stated in Figure 2.4. The lower domain is where the real world interactions takes place (sensing and/or actuation) while the other domain represents everything network related. For a better understanding, the next paragraphs will give a brief description of each entity represented in the Figure 2.4.

M2M Device. There two types of devices. One that doesn't have a local SCL, but can communicate with one, using them as a proxy to connect with the NSCL. The other type is a device powerful enough to have a local SCL (DSCL),therefore being able to connect directly with the NSCL in the network domain. It does not need to be intermediated by a gateway, plus it can intermediate the connection between the network domain and other less capable devices that cannot connect directly. Finally, both types are capable to run DAs (Device

Applications), that can exploit services and provided functionalities through the interaction with the DSCL.

Legacy Device. This is a device without a local SCL and noncomplying with the standard. Consequently it needs a gateway or device with a SCL to intermediate its connections.

M2M Gateway. This entity acts as a gateway/proxy to connect less capable devices to the NSCL. It has a GSCL as capability layer and can also run M2M applications (Gateways Applications (GAs)). Comparing to M2M devices with local SCL, the only difference, is that the gateway is suppose to be more resource capable to able to run more demanding services and to support more connected devices.

Access and Core Network. These represent the physical networks that connect the domains.

M2M Service Capabilities (NSCL). This entity provides an abstraction over the Core and Access Networks, attaining a service layer between the Device and Gateway Domain and the M2M Applications. It has the NSCL as capability layer, that acts as a platform to connect M2M applications (Network Applications (NAs)).

M2M Applications (NA). These are the applications that connect through the NSCL interface to interact with the lower domain.

2.5.3 AVAILABLE IMPLEMENTATIONS

Acknowledging the advantages of the ETSI M2M standard, many companies started developing their own compliant solutions. This section will describe some of the most relevant solutions available.

2.5.3.1 COCOON

Cocoon it's an open source project, ETSI M2M compliant gateway and developed by Actility since 2011. As objective, they pretend to "turn the Internet of Things dream into a reality"[1], they want to provide means to interconnect M2M application developers and hardware manufacturers building ETSI M2M compliant gateways, and therefore push the IoT dissemination.

Cocoon takes advantage of the RESTful APIs of ETSI M2M to provide an abstract layer between the applications and the underlying hardware. It also leverages the OSGi embedded

[1]http://cocoon.actility.com/documentation/ongv2/FAQ

Java environment, making application components hardware independent as well as providing multiple services such as: data containers, data logging, subscriptions and notifications. An overview of the implementation can be seen in Figure 2.5.

Actility as also developed and deployed other IoT related solutions, in later 2003 they launched an app store, the ThingPark Store[2] in order to facilitate the dissemination of Internet of Things applications.

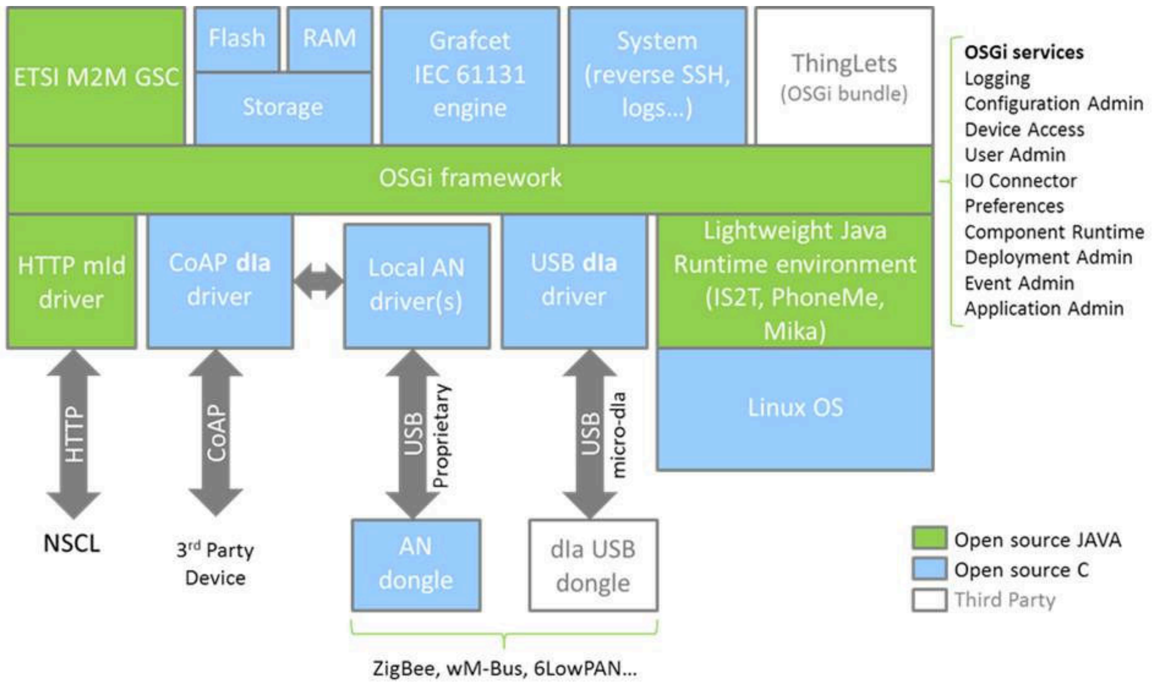


Figure 2.5: Cocoon OSGi environment [13]

2.5.3.2 OM2M

OM2M it's an open source implementation of the ETSI M2M standard initiated by the *Laboratoire d'analyse et d'architecture des systèmes*[3]. It facilitates service deployment by providing a development framework that provides abstraction over the underlying network. The project also aims, to be the basis for the future implementation of the oneM2M standard.

OM2M was implemented in Java and proposes a modular architecture that runs on top of an OSGi layer, which per se, brings a bundle of service but also makes it highly extensible via a plugin system. The Figure 2.6 represents an overview of the project architecture.

[2]http://store.thingpark.com/

[3]https://www.laas.fr/public/fr

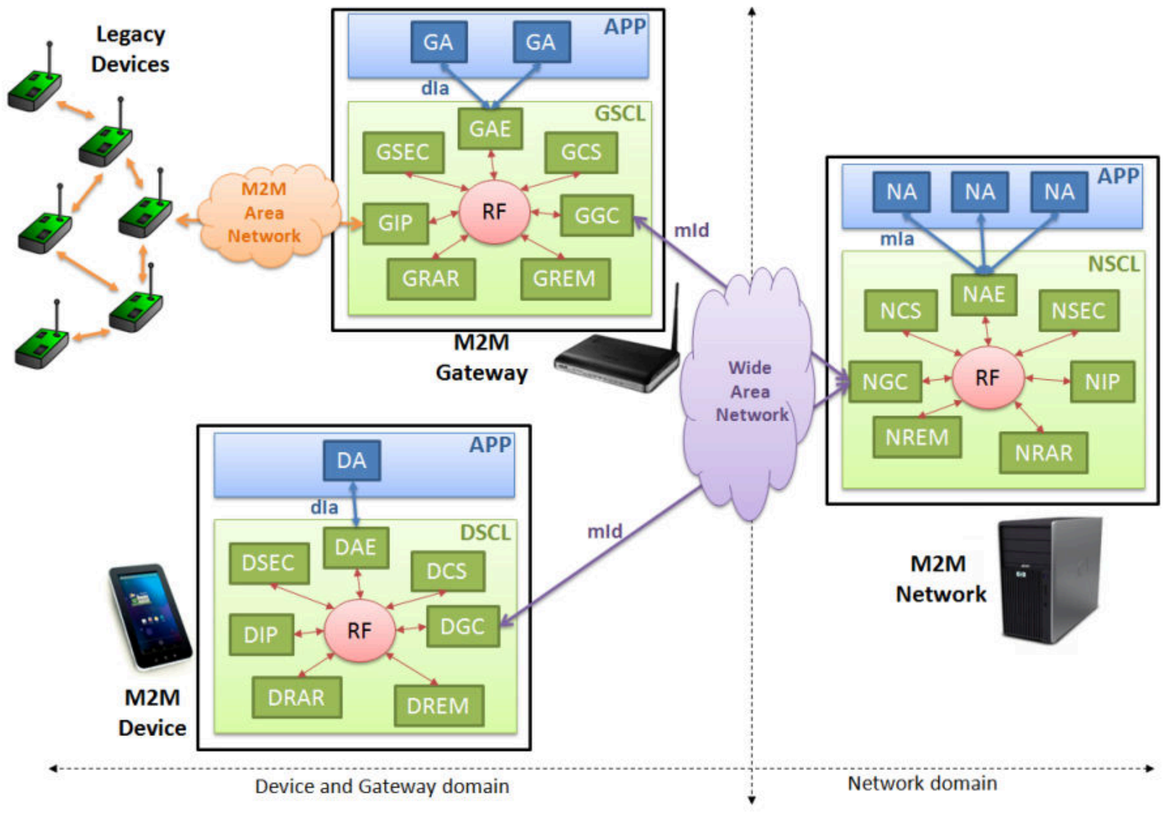


Figure 2.6: OM2M Architecture [14]

2.6 IOT PLATFORMS (RELATED WORK)

The evolution of M2M middleware solutions, brought the need for wider software platforms, that could handle M2M communication as well as service/application deployment, integrating all in the context of the IoT. Therefore, new business models started to arise, pushing companies to create new solutions for IoT scenarios, reaching both device vendors and application developers.

At this moment, there are dozens of IoT Plaforms, both open source and commercial. This section will give a brief overview on some of those solutions.

2.6.1 CARRIOTS

Carriots is a commercial Platform as a Service (PaaS) App Engine aimed at Makers and Developers of IoT and M2M products and projects.

For application development, the platform offers the Carriots App Engine. It supports Groovy as programming language and eases application development by providing a simple development environment and worry-free hosting. It also provides Rule Management and SDK for controlling business logic.

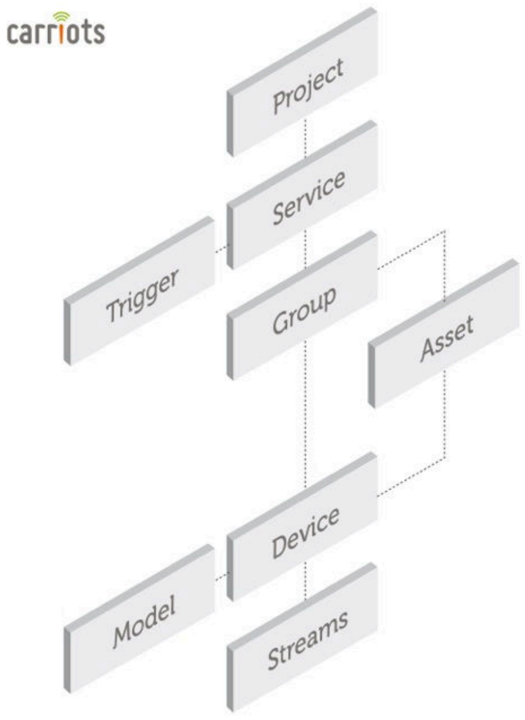


Figure 2.7: Carriots Hierarchy [15]

As for devices, Carriots offers a device management module for: status check, configuration change and firmware upgrades. It also provides Data Collection and Data Storage services for data related operations.

Project Management is also possible, the project concept can cluster multiple hierarchical components as shown in Figure 2.7, in which access management between customers and devices can be controlled.

2.6.2 OPENMTC PLATFORM

Open Machine Type Communication (OpenMTC) is a cooperative development of Fraunhofer FOKUS and Technische Universität Berlin (TUB). It is a platform based in the latest standards and provides a realistic cross-domain horizontal M2M platform implementation [16].

The OpenMTC implemented an ETSI M2M compliant M2M middleware. It is compose by two SCLs, a GSCL and a NSCL. The GSCL supports multiple M2M technologies and communication protocols while the NSCL stands as a cloud-based M2M platform for data aggregation and storage.

Interaction with telecommunication infrastructures is also possible by supporting IP Multimedia Subsystem (IMS) and Evolved Packet Core (EPC). Enabling the usage of IMS

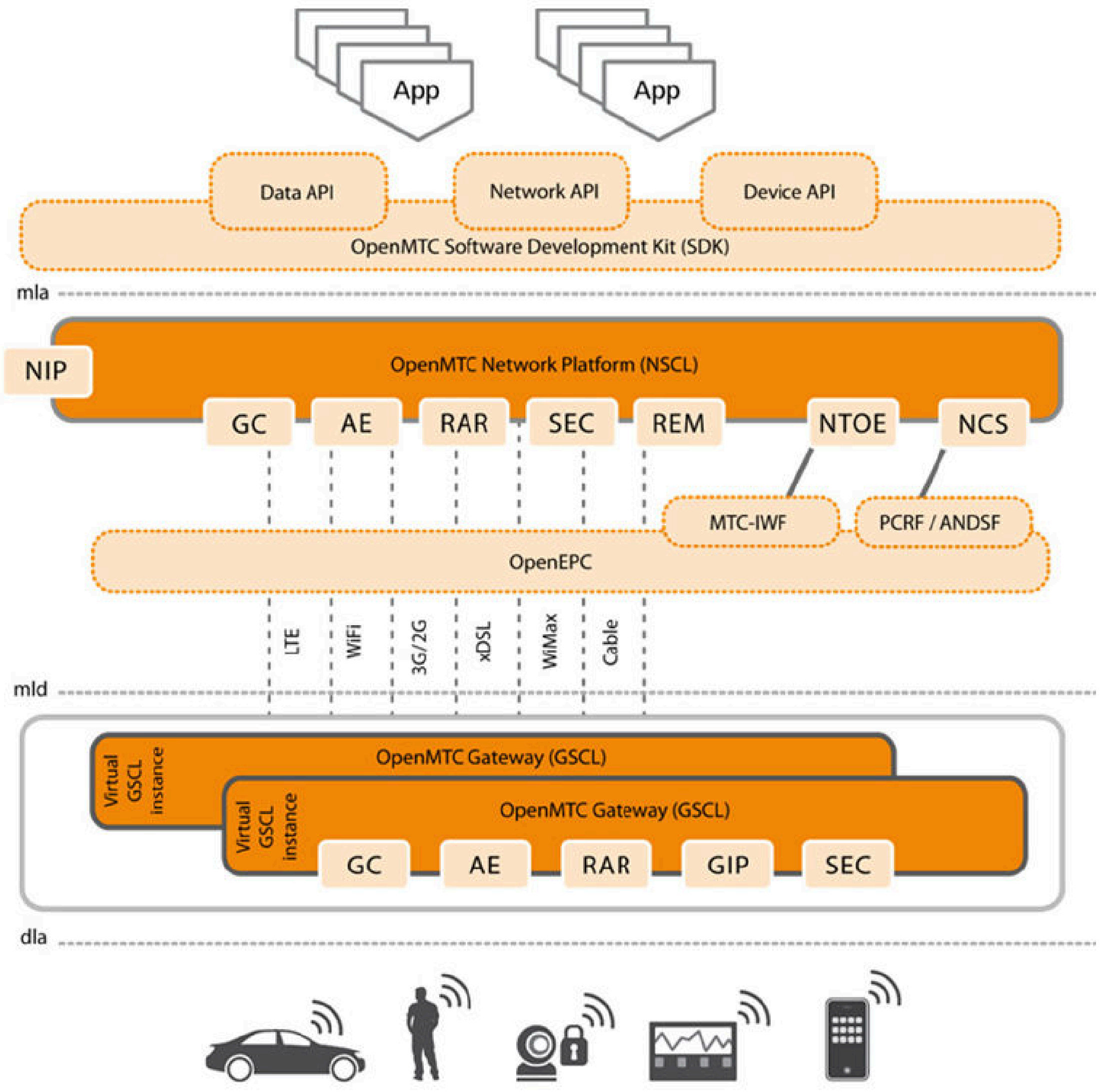


Figure 2.8: OpenMTC Platform Architecture [16]

for M2M applications by translating the information exchanged from sensors and devices to Session Initiation Protocol (SIP). Thus, potentiating M2M applications to rely on the security, reliability of existing deployments of IMS as well as Quality of Service (QoS) capabilities brought by the EPC.

To ease and improve the development of M2M applications, the OpenMTC provides an SDK, making available core assets and service capabilities to third party developers.

The Figure 2.8 shows an overall representation of the OpenMTC Platform architecture, giving a better understanding on how everything fits together.

2.6.3 XIVELY

Formely known as Cosm and Pachube, Xively[4] is a division of LogMeIn Inc, and it is a Platform as a Service (PaaS) for the IoT.

Xively, defines itself as a "Public Cloud specifically built for the Internet of Things"[17]. It enables the deployment of devices, applications and services, as illustrated in Figure 2.9.

A list of libraries for multiple languages and platforms are provided by the Xively solution to properly and easily deploy/connect a device to the platform, so it could send data, as well as to benefit from in-house and user services. Once integrated, services for device management and provisioning can be access from within the Xively platform, or through user applications using the provided SDKs and platform APIs.

The Xively platform, also offers a high performance, flexible data service, with time-series database to store and retrieve data, as well as a triggering system, enabling advanced actions to execute across any connected device, application or service. This data, generated by user devices and applications, can be used privately, as well as be shared through external services like Twitter, Facebook or user specified services.

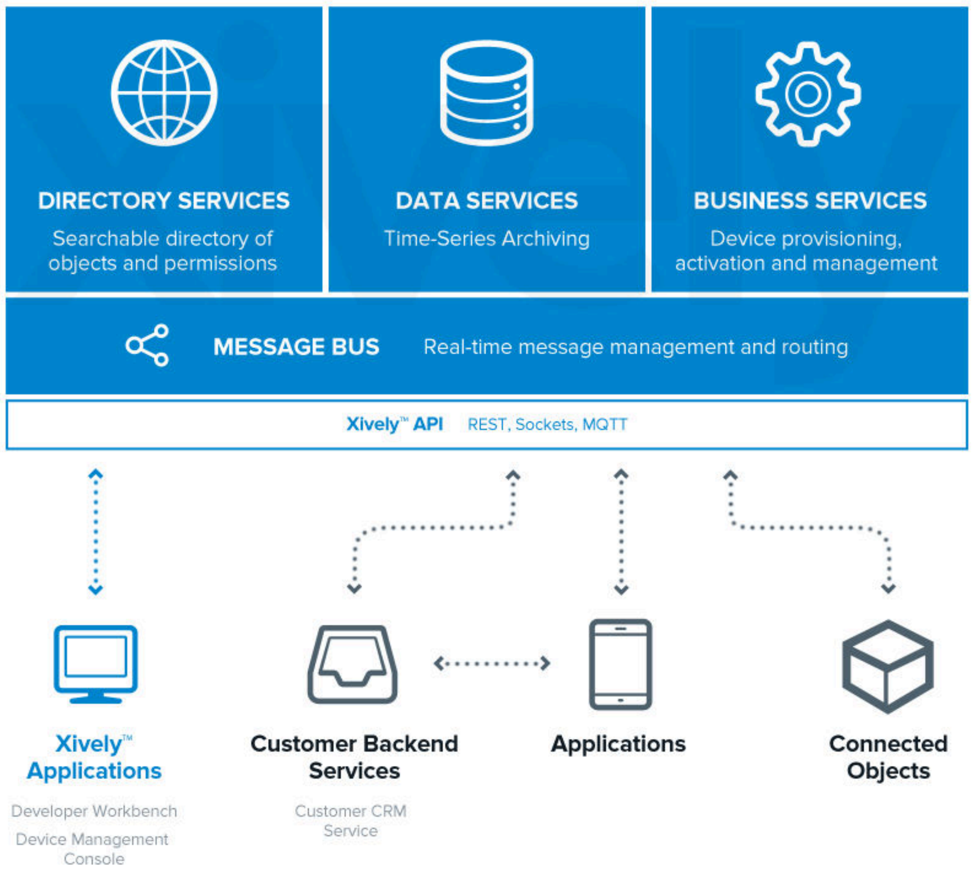


Figure 2.9: Xively Cloud Services^{TM}[17]

[4]https://xively.com/

2.7 IDENTITY AND ACCESS MANAGEMENT

The growth of organizations and their increasing and complex systems, bring old and new concerns regarding the management of employee identities and its access control. IAM stands as a collection of processes and technologies to manage those digital identities and the resource access provided through them. It can be defined in two core components - identity management and access management [18].

The identity management refers to the management of digital identities and profiles. It establishes unique identities and authentication credentials, enabling workflows to correctly identify identities within a system.

The access management or entitlements management, designate the processes and technology to control access to information for a specific identity. The access rights and privileges for an authenticated identity, are specified as a set of attributes (entitlements).

This section will explain concepts and describe important technologies existing today in the context of Identity management (IdM).

2.7.1 OPENID CONNECT

OpenID Connect[5] is an identity layer built on top of the OAuth 2.0 protocol, it is an evolution from OpenID 2.0 with a friendlier API and usable by natives and mobile applications. It provides Single Sign-on (SSO) capabilities as well as basic profile information exchange, all in an interoperable and REST-like manner.

It was on February 26, 2014, that OpenID Connect achieved the final specification (v1.0), the Figure 2.10 represents an high level view of the its components. These components, are specifications of functionalities, they explain in detail how to accomplish that functionality. OpenID Connect offers capabilities like Dynamic Client Registration and Discovery, the latter can dynamically discover information about OpenID Providers, therefore solving the "nascar problem"[19].

The core component of OpenId Connect[21], specifies how authentication was built on top of OAuth 2.0 and how to use *claims* to exchange end-user information. It specifies three different authentication flows, these flows describe workflows between multiple entities, in order to retrieve secure tokens. These tokens, can identify clients and attest them as trustworthy to access secure end-user information. The next subsections will describe and explain each authentication flow.

[5]http://openid.net/connect/

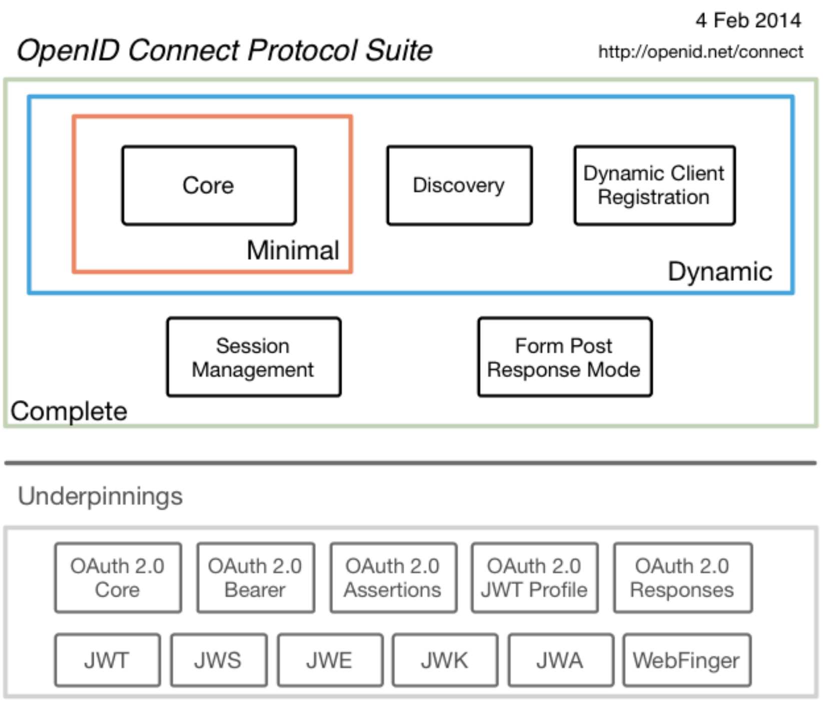


Figure 2.10: OpenID Connect Protocol Suite[20]

2.7.1.1 AUTHORIZATION CODE FLOW

This represents the complete authentication flow, using all endpoints, and therefore allowing a more secure workflow. It presents a separation of concerns, as the user authentication and consent is done between Authorization Server and End User, the tokens are generated only in the Token Endpoint and the user information is managed and retrieved at the UserInfo Endpoint. Figure 2.11 represents a diagram with this authentication flow and each step is explained in the following enumerated list:

1. Relying Party (the Client), prepares an Authorization Request containing the desired request parameters;
2. Relying Party sends the request to the Authorization Server;
3. The Authorization Server authenticates the End User;
4. Authorization Server obtains the End User consent;
5. Authorization redirects the End User back to the Relying Party with an Authorization Code;
6. The Relying Party requests the Token Endpoint using the Authorization Code;
7. The Relaying Party receives the an ID Token and Access Token;

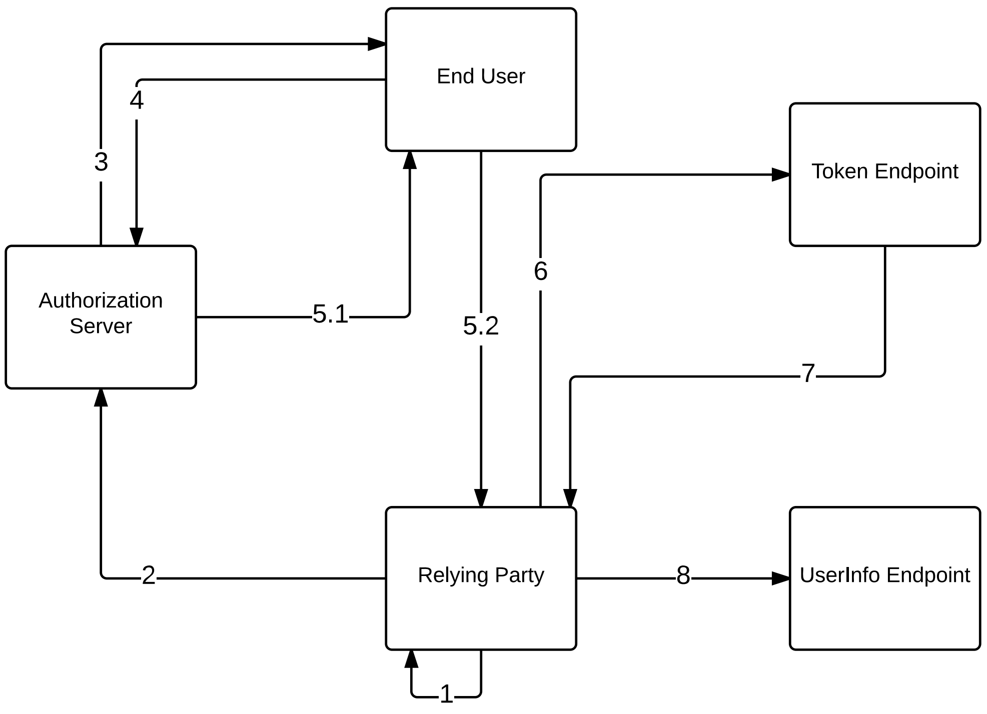


Figure 2.11: Authorization Code Flow[21]

8. The Relaying party validates the ID Token and requests End User information at the UserInfo Endpoint.

2.7.1.2 IMPLICIT FLOW

The Implicit Flow doesn't require the Token Endpoint, since all tokens are returned from the Authorization Endpoint. This authentication flow is simpler than the Authorization Code Flow, but can expose access information to the End User browser user agent or malicious browser extensions. Figure 2.12 represents a diagram with this authentication flow and each step is explained in the following enumerated list:

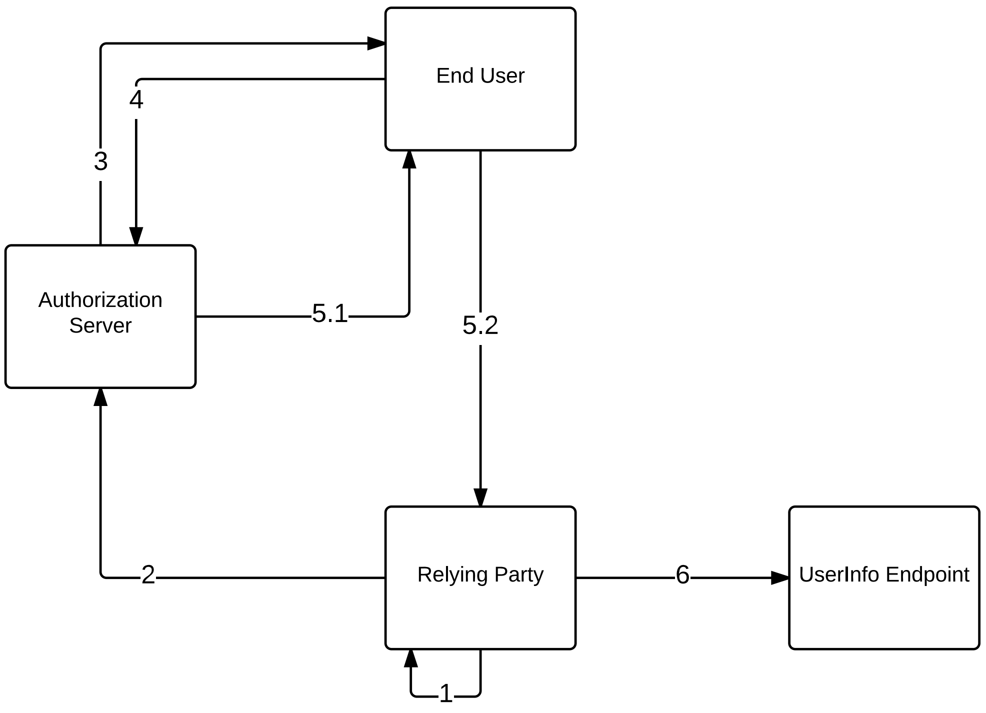


Figure 2.12: Implicit Flow[21]

1. Relying Party (the Client), prepares an Authorization Request containing the desired request parameters;
2. Relying Party sends the request to the Authorization Server;
3. The Authorization Server authenticates the End User;
4. Authorization Server obtains the End User consent;
5. Authorization redirects the End User back to the Relying Party with an ID Token and Access Token (if requested);

6. The Relaying party validates the ID Token and requests End User information at the UserInfo Endpoint.

2.7.1.3 HYBRID FLOW

OpenID Connect support a third authentication flow, the Hybrid Flow, this authentication flow can retrieve tokens from the Authorization Endpoint and from the Token Endpoint, i.e. it can act as a normal Authorization Code Flow or simplify by receiving the tokens from the Authorization Endpoint like the Implicit Flow. This can be done by specifying the *response_type* (OAuth 2.0 Response Type) value in the request message to the Authorization Endpoint. No diagram is necessary to explain, since it the same flow of the Authorization Code Flow, but with the capability of setting the request to Token Endpoint as optional.

2.7.2 SAML 2.0

The Security Assertion Markup Language (SAML) is an OASIS standard [22] with a defined XML-based framework for exchanging security information between online business partners.

SAML V1.0 was announced in November 2002 and it took one year to receive a minor upgrade (V1.1) in September 2003. The new version brought some new features and some minor corrections [23] but it was in March 2005 that SAML received a major upgrade V2.0. This version was not retro-compatible but came with multiple improvements and support to new technologies.

There are three main components specified in SAML: assertions, protocols and bindings. This components, when put together, can potentiate multiple use cases. The Figure 2.13 represents these components, where the Profiles are the representation of the component combination necessary to support a defined use case. There are two other components present in the Figure, the Metadata and Authentication Context. The first defines how to express and share configuration information between SAML parties, while the second refers to authentication details that some times are needed, to inform about the authentication performed or to specify authentication requirements between Service Provider (SP)[6] and Identity Provider (IdP)[7].

This section aims to give an overview to SAML 2.0, describing its main components and giving a more detailed analysis over some key use cases.

Roles. There are three roles in SAML: the principal (typically a user), the IdP and the SP. In a simple use case, the principal requests a service from the SP, the SP requests and

[6]Defined in Glossary
[7]Defined in Glossary

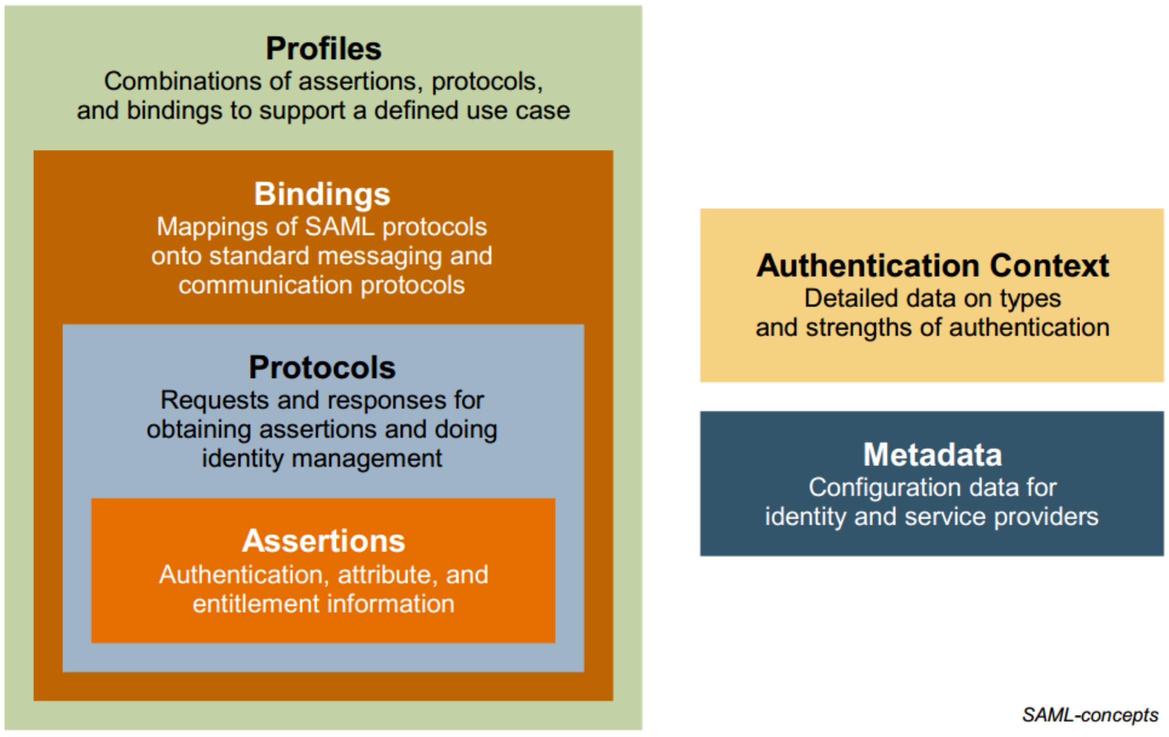

Profiles
Combinations of assertions, protocols, and bindings to support a defined use case
Bindings
Mappings of SAML protocols onto standard messaging and communication protocols
Protocols
Requests and responses for obtaining assertions and doing identity management
Assertions
Authentication, attribute, and entitlement information
Authentication Context
Detailed data on types and strengths of authentication
Metadata
Configuration data for identity and service providers
SAML-concepts

Figure 2.13: Basic SAML Concepts[22]

receives an identity assertion from the IdP and then make an access control decision, to allow or not access to the principal.

Assertions. An assertion could be seen as a package of information issued by a SAML authority. This package could have multiple statements and they are all associated with a subject. There are three different kinds of statements: Authentication Assertion and it represents the subject authenticated and information about that authentication; Attribute Assertion that represents the attributes associated with the subject; Authorization Decision Assertion which is the decision about whether or not the subject can access a desired attribute.

Protocols. SAML defines a number of generalized request/response protocols, in which describes how assertions and other SAML elements are packaged. Thus, providing guidelines to SAML entities for producing and consuming this elements.

There are 6 protocols defined in SAML 2.0[24]:

- Assertion Query and Request Protocol;
- Authentication Request Protocol;
- Artifact Resolution Protocol;
- Name Identifier Management Protocol;

- Single Logout Protocol;
- Name Identifier Mapping Protocol.

Bindings. The SAML bindings defines, how exactly, the various SAML protocol messages can be mapped onto standard messaging formats and communication protocols.

There are 6 bindings defined in SAML 2.0[25]:

- SAML SOAP Binding;
- Reverse SOAP (PAOS) Binding;
- HTTP Redirect Binding;
- HTTP POST Binding;
- HTTP Artifact Binding;
- SAML URI Binding.

Profiles. The SAML profiles defines the combination of assertions, protocols and bindings in order to provide greater interoperability for usage scenarios.

The list of SAML 2.0 profiles goes as follows[26]:

- SSO[8] Profiles of SAML:
  - Web Browser SSO Profile;
  - Enhanced Client or Proxy (ECP) Profile;
  - Identity Provider Discovery Profile;
  - Single Logout Profile;
  - Name Identifier Management Profile.
- Artifact Resolution Profile;
- Assertion Query/Request Profile;
- Name Identifier Mapping Profile;
- SAML Attribute Profiles:
  - Basic Attribute Profile;
  - X.500/LDAP Attribute Profile;
  - UUID Attribute Profile;
  - DCE PAC Attribute Profile;
  - XACML Attribute Profile.

There are many profiles, as well as different types of profiles. Thus, for brevity and regarding the context of this dissertation, only a couple of profiles will be analyzed, being both of them SSO profiles.

[8]Defined in Glossary

Web Browser SSO Profile. This profile defines the necessary SAML messages and bindings to support a web SSO use case. It provides a wide variety of options, as the message flow can be IdP-initiated or SP-initiated, as well as which bindings are used to deliver the messages between the two. Therefore, enabling many combinations between message flows and bindings. The Figure 2.14 stands for a SP-initiated SSO using a Redirect Binding for the SP-to-IdP <AuthRequest> message and a POST Binding for the IdP-to-SP <Response> message (The other combinations possible will not be addressed).

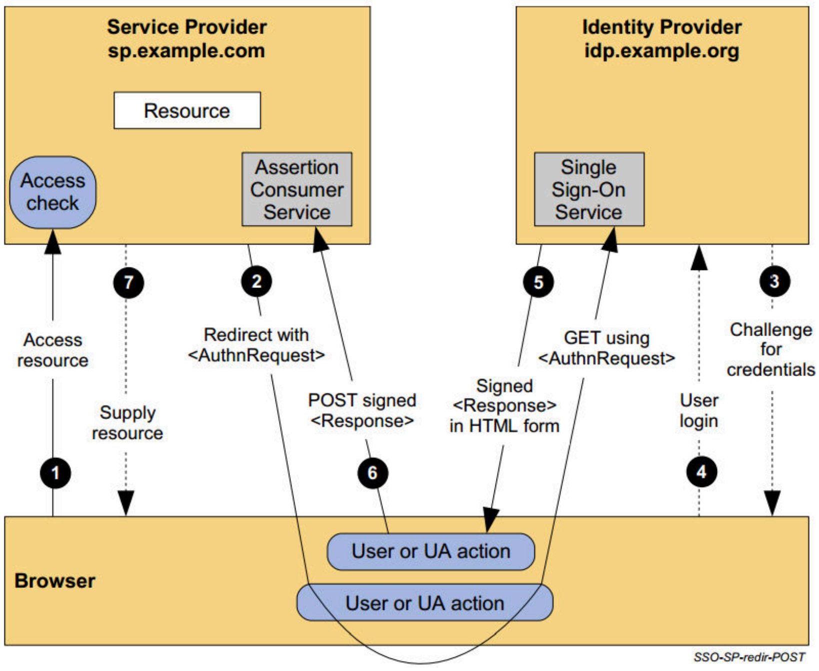


Figure 2.14: SP-Initiated SSO with Redirect and POST Bindings[22]

Enhanced Client or Proxy (ECP) Profile. This profile takes into account enhanced client devices and proxy servers. They could be clients with more capabilities than those of a browser, thus, allowing them to have a more active participation in the IdP discovery as well as the message flow. Another use case, is using a proxy server, providing a gateway the less capable devices, for example a WAP gateway for a cellphone (not smartphone). The Figure 2.15 illustrates an use case using Reverse SOAP (PAOS) as Binding, this will take advantage of the SOAP headers and SOAP bodies to transport the messages between SP and IdP.

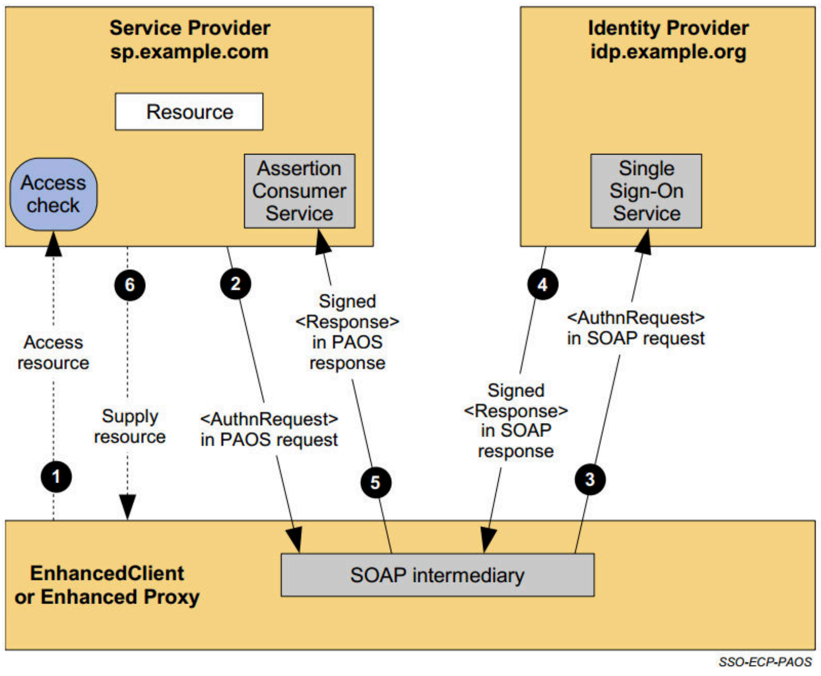

Service Provider
sp.example.com
Resource
Access check
Assertion Consumer Service
Identity Provider
idp.example.org
Single Sign-On Service
1
Access resource
6
Supply resource
2
<AuthnRequest> in PAOS request
Signed <Response> in PAOS response
5
4
Signed <Response> in SOAP response
<AuthnRequest> in SOAP request
3
EnhancedClient or Enhanced Proxy
SOAP intermediary
SSO-ECP-PAOS

Figure 2.15: SSO Using ECP with the PAOS Binding[22]

2.7.3 WS-SECURITY

Web Services Security (WS-Security) is an OASIS Standard Specification and was created on April 5 2002 [27], with the purpose of enhancing the SOAP messaging to provide message integrity and confidentiality.

This specification was designed to act as the basis for securing web services within a wide variety of security models. Thus, to be flexible enough to support multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. Furthermore, WSS describes how to encode binary security tokens, specifically, how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys.

WSS specifies three main mechanisms: ability to send security tokens as part of the message, message integrity, and message confidentiality. By themselves these mechanisms do not provide a complete security solution for Web services, but instead, they can be a building block to be used in coexistence with other Web service extensions and protocols to shelter a wide variety of security technologies and security models.

As an example this mechanisms can be used for signing and encrypting a message or part of a message and providing a security token or token path associated with the keys used for signing and encryption.

2.7.4 WS-TRUST

WS-Trust is an OASIS Standard Specification created in 2007 and is currently on version 1.4 [28]. It is an extension to WS-Security that aims to provide a framework for requesting and issuing security tokens, as well to establish, assess the presence of, and broker trust relationships between participants in a secure message exchange.

This specification introduces some new elements and concepts, as: the concept of Security Token Service (STS)[9]; message formats to request security tokens as well as the responses for those messages; key exchange mechanisms.

2.7.5 XACML

The eXtensible Access Control Markup Language (XACML) is an OASIS standard [29], it defines a core XML schema for representing authorization and entitlement policies. The version 1.0 was announced in February 2003 and since then much work was done, currently its on version 3.0 released on August 2010 and updated on January 2013.

XACML standard addresses fine grained control of authorization, it defines a police language to describe general access control requirements. An access control decision request/responce language is also defined, this enables queries against policies to verify if an action is allowed or not. XACML 3.0 brought delegation, in which authority can be delegated to another entity without the need to modify the root policy and therefore support decentralized administration of access policies.

As an example and for a better understanding, the Figure 2.16 illustrates a complete setup for an XACML use case. There are four main components present in a complete XACML workflow[30]:

- **Policy Enforcement Point (PEP):** The system entity that performs the access control, by making decision requests and enforce authorization decisions;
- **Policy Decision Point (PDP):** The system entity responsible for evaluating policies and generating authorization decisions;
- **Policy Information Point (PIP):** The system entity that acts as source of attributes values;
- **Policy Administration Point (PAP):** The system entity responsible for creating the policies.

The illustrated workflow, represents the necessary steps taken, to correctly assess a request from a client. As an overall explanation, upon the client request, the PEP entity will request the authorization system, if the client has the necessary permissions to access those resources. Hence, the PDP and PIP entities will exchange information to identify those same

[9]Defined in Glossary

resources, once identified the PDP will process that information with the previous defined policies written by the PAP entity. Finally, according to the decision processed, a response to the PEP will be sent, informing if the client has enough permissions or not.

The division of the authorization system in multiple entities and its powerful policy language and engine, makes the XACML very flexible and enabling support for a wide variety of use cases.

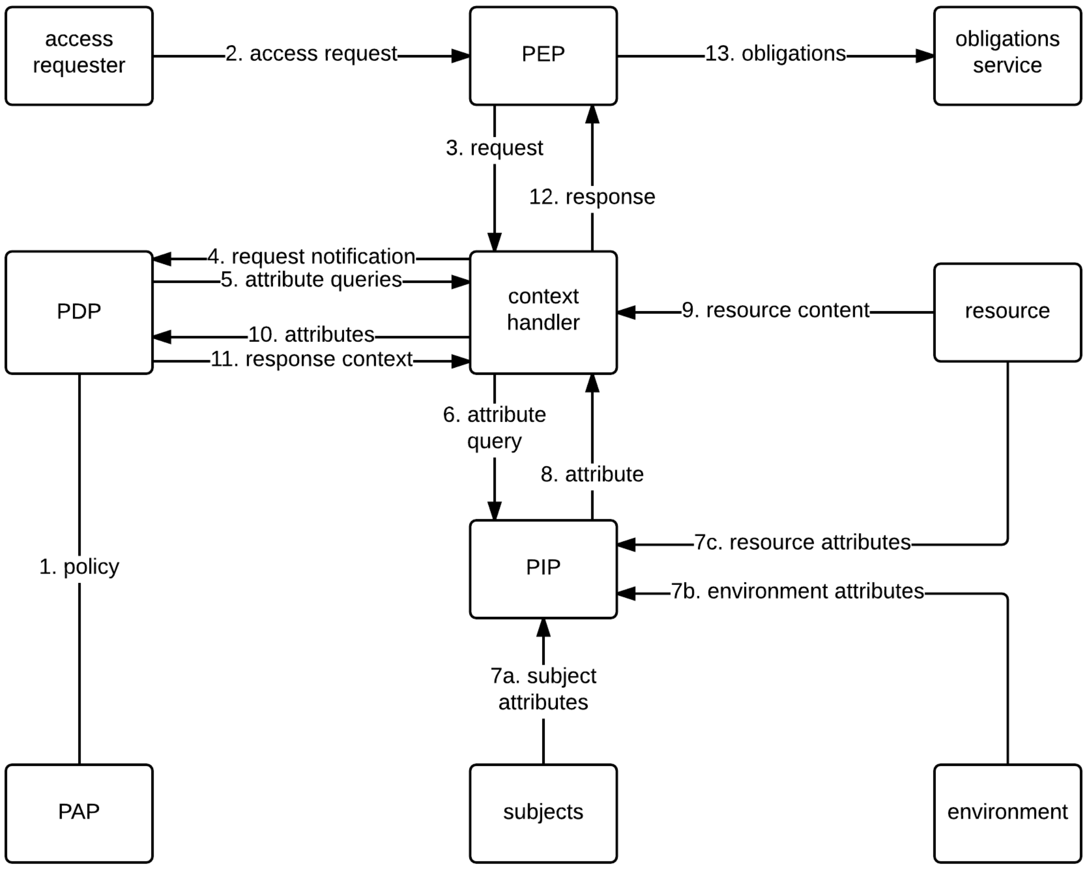


Figure 2.16: XACML Data-flow diagram[30]

2.7.6 SOFTWARE

There are many applications that provide complete or partial IAM, some offer a complete solution ranging from full identity management, multiple support to existing SSO implementations and fine and course grained access control, while other solutions only focus on a specific component.

This subsection will introduce two IAM solutions.

2.7.6.1 WSO2 IDENTITY SERVER

WSO2 Identity Server[10] provides secure identity management for enterprise web applications, as well as services and APIs, reducing identity provisioning time and guaranteeing secure online interactions. It is an open source solution and was built on top of WSO2 Carbon[11], enabling easy customization and extension through its architecture. Figure 2.17 depicts the WSO2 Identity Server architecture, representing its functionalities divided in components. The next paragraphs will give an overview description on some of the principal components, focusing on authentication and authorization.

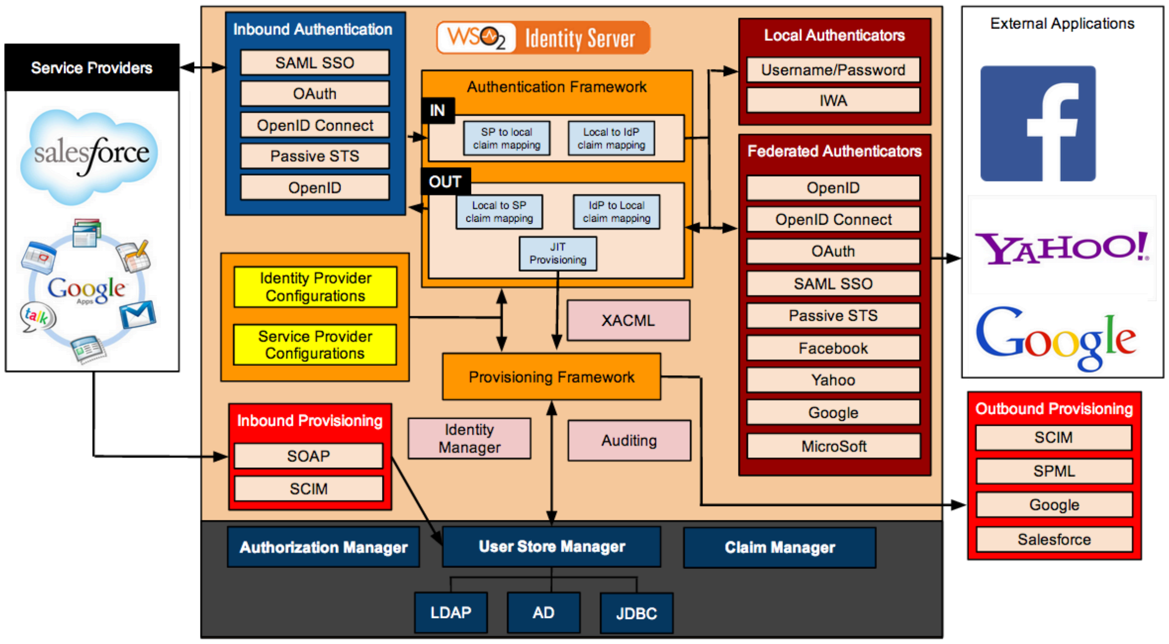


Figure 2.17: WSO2 Identity Server 5.0.0 Architecture[31]

Authentication. WSO2 Identity Server, offers several *out of the box* authentication mechanisms (Local and Federated Authenticators components), this facilitates the integration with applications and services who wants to easily delegate their authentication to a solution like the Identity Server. Nevertheless, it is also possible to rely on the Identity Server only as a standard identity provider (Inbound Authentication component), and therefore, register applications or services as service providers.

There are a handful of supported authentication mechanisms, both in Inbound Authentication and Local/Fedearated Authenticators, although, the Inbound relies only on federated mechanisms like SAML SSO, OpenID and OpenID Connect, Passive STS (WS-Trust STS), as well as OAuth (despite OAuth being associated with authorization). The Federated Au-

[10]http://wso2.com/products/identity-server/
[11]http://wso2.com/products/carbon/

thenticators also provide support for login systems from well known companies like *Facebook*, *Yahoo*, *Google* and *Microsoft*.

Authorization. WSO2 Identity Server contains an advanced entitlement auditing and management system. It supports claim-based access control via WS-Trust, OpenID and OpenID Connect as well as OAuth. Support for Role-based Access Control (RBAC) and fine grained policy-based access control via XACML is also available.

Regarding policies, the WSO2 Identity Server offers a friendly user interface for policy creation and management, easing and accelerating the integration between applications or services and the Identity Server for XACML access control. It also supports the registration of multiple PIPs and multiple PDPs for policy distribution.

2.7.6.2 JBOSS PICKETLINK

JBoss PicketLink stands for a "Simplified Security and Identity management for Java Applications"[12]. It is an umbrella project for security and identity management, therefore, encompasses multiple subprojects/modules to tackle different security and identity scenarios. The following paragraphs will explain and describe the technologies supported by this solution.

Java EE Application Security. This module provides mechanisms to integrate authentication and authorization with Java EE applications through Contexts and Dependency Injection (CDI), JavaServer Faces (JSF) bindings and platform configurations (focused on JBoss Enterprise Application Platform). It enables for example, multiple HTTP authentication mechanisms, Re-Captcha and two factor authentication scenarios. Authentication using REST endpoints is also supported using Java API for RESTful Web Services (JAX-RS) Authentication Endpoint. Furthermore, to ease development and integration with login systems from social networks like *Facebook*, *Twitter* and *Google*, it provides snippets, configurations and CDI references for a correct authentication.

For authorization, PicketLink offers support for Access Control List (ACL), Java Persistence API (JPA), Lightweight Directory Access Protocol (LDAP) and RBAC based authorization scenarios, as well as the integration with frameworks like Drools[13].

Identity Management. This is a fundamental module of PicketLink, being a base for other modules to implement their extended features. It provides a rich and extensible API for managing identities, such as users, groups and roles. This provides means to connect to LDAP servers and enables the creation of multi-tenancy architectures.

[12]http://picketlink.org/
[13]http://www.drools.org/

Federation. PicketLink offers configurable services, to handle federation scenarios like SSO and identity propagation. It provides SAML 1.1 and 2.0 support as well as OpenID support for SSO scenarios. For identity propagation, a WS-Trust STS is provided, this enables complex authentication scenarios between multiple entities.

Regarding authorization, and to enable fine and course grained authorization, PicketLink provides support and services for OAuth2 and XACML(v2) workflows.

2.8 SERVICE-ORIENTED ARCHITECTURE

SOA is a software design and a software architecture design pattern, it provides application functionality as services to other applications and it is also independent from vendor or technology.

Before SOA, in-house solutions were typically composed by multiple applications, this tended to create code redundancy and duplication of functionalities. SOA brought architecture design around services rather than applications, presenting services as small units of software with a specific function that can be reused by any application, thus, reducing the previous problems of redundancy and duplication. Furthermore, by distributing functionalities by service, enables the creation of more complex applications through orchestration.

Despite the advantages of SOA, since its beginnings (early 2000s), it has been the subject of much controversy, it received both praises as a modern and agile approach to software development as well as considered a colossal waste of time and money[32]. The main causes for this problematic, tends to the design principals and methodologies taken by the vendors for the implementation of their solutions.

When adopting SOA, there are two possible approaches[32] , top-down and bottom-up. The first generally starts by launching a wide SOA initiative englobing most of the organization, often binded to a proprietary SOA stack from a big vendor. This entails high upfront costs, ranging from licensing, the need to exchange software platforms and hardware to meet system requirements, as well as big learning curve for in-house developers. Hence, this being the approach with the less successful results. In the other hand, the bottom-up approach benefits an incremental adoption instead of an extensive reengineering. A typical scenario, is to start off with a standalone Enterprise Service Bus (ESB), acting as core component and providing a stable base for a steady incremental adoption. This prevents vendor lock-in, since most ESBs are built according to open standards, enabling interoperation and easing the learning curve. That being said, the bottom-up approach represent a better Return On Investment (ROI) and a less stressful integration.

Today, SOA is still relevant, while there were disbeliefs in its adoption, the ones that succeed, achieved a reliable infrastructure capable of scaling in today's demanding market. The the bottom-up approach is already explained and proven the best option for future adoptions.

This section will present two SOA technologies, the ESB and the SCA. The latter will also have an analyses on some of the current implementations available.

2.8.1 SERVICE DISAMBIGUATION

The term "service" is much used in SOA and SOA related solutions, but can be misleading if not introduced in the proper context. The authors in [33], re-examined concepts and terminology used when describing SOA and ESB solutions, and proposed a more refined terminology to prevent misconceptions. Concerning the term "service", the following paragraphs present three new terms to improve understanding and reduce confusion when describing SOA solutions.

Governed Service. Refers to a business task but in a vaguely manner, i.e., describe a service without being precise if talking about the specification or implementation. For example, "We have a suite of governed services for administering customers' accounts".

Service specification. It is formal set of characteristics to describe the interaction with a service, it specifies the business task achieved, its technical requirements to a correct interaction, as well as Quality of Service (QoS) for that interaction.

Service Realization. This refers to the physical implementation of a service specification. The common use of the term "service" normally refers to this realization. It represents the actual implementation of the service and how its business task is processed.

The Figure 2.18 illustrates an abstract view in a service mediation, using the new terminology.

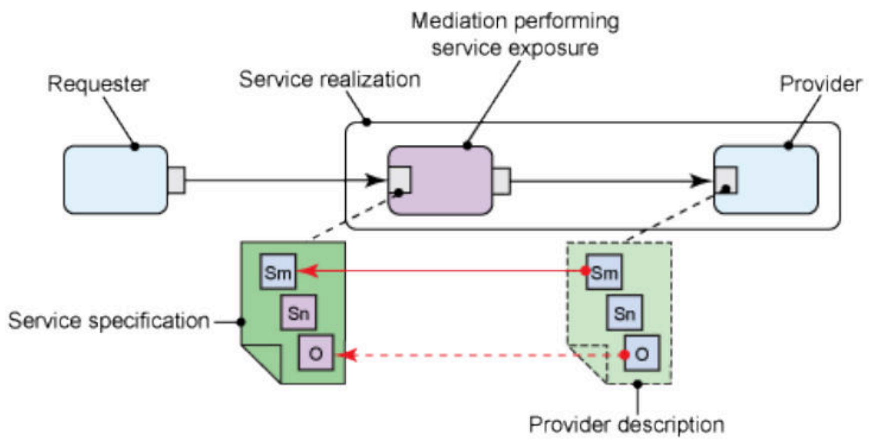


Figure 2.18: Mediating service interaction [33]

2.8.2 ENTERPRISE SERVICE BUS

ESB is an architecture pattern that supplies loosely coupled connectivity between service requesters and service providers in a service-oriented fashion. Therefore, providing a clean separation of concerns, enabling flexibility and agility in both business processes and IT systems.

While the term "Enterprise Service Bus" was first introduced by Roy Schulte from the Gartner Group 2002, more than a decade after, an accepted definition for this term is yet to be firmly established. A likely cause for this, is the lack of a global standard for the concepts or implementations. Thus, as a consequence, there are different definition depending on the manufacturer or source [34].

Many of the ESB solutions existing today, are based on existing architectures, ranging from Enterprise Application Integration (EAI), Message-Oriented Middleware (MOM), Application Servers, Business Process Management Systems (BPMS) as well as many others architectures that, in a way or another, handle service exposure and mediation.

While there are many available ESB solutions, and each one supporting a different variety of features, there are at least, a common base of features that all support[34] - Routing, Protocol bridging and Message transformation. Regarding the Routing feature, an ESB must be able to route messages to services endpoints, this routing is based on pre-defined policies, such as message attributes or content, identity, time of day or system load balancing. It also provides, a virtualization layer in order to enable versioning, location, binding and dynamic service selection. The ESB is also capable to mediate between multiple protocols - SOAP and REST for example - abstracting the service implementation from its exposure, this stands as the Protocol bridging feature. Finally, regarding the Message transformation feature, it provides capabilities for XML-based data processing, thus allowing validation, aggregation, filtering and XML message content transformation. Besides the common base features, Service hosting and Resource adapter are also frequent to find in ESB solutions.

2.8.3 SERVICE COMPONENT ARCHITECTURE

SCA is a set of specifications which describe a model for building SOA-based applications and systems. It was designed in a conjoint effort between major software vendors including IBM and Oracle, releasing version 1.0 of the specification on March 21, 2007. It was build on open standards to allow vendor implementation and support.

SCA abstract the middleware programming model dependencies from business logic. This removes the inherent middleware complexity from application developers, allowing them to focus on writing business logic. Figure 2.19 illustrates a generic high-level architecture for an SCA application, showing a composition of two implementations.

The next subsections will describe the SCA Assembly Model, which consists of a series of artifacts to specify and represent how an SCA application is designed and implemented.

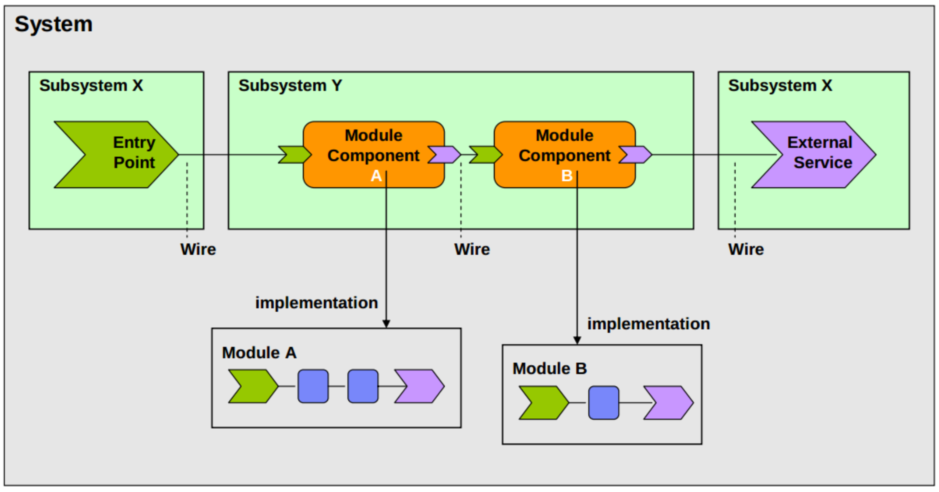


Figure 2.19: SCA example [35]

2.8.3.1 COMPOSITES

A composite is a logic construct, it can contain one or more components, these components can run in a single process on a singe computer or be distributed across multiple processes on multiple computers. The SCA composite defines a complete application, describing combinations of components, how they are connected, references they use and services they promote. This is normally described in an XML-based associated configuration file, using a format called Service Component Definition Language (SCDL).

Figure 2.20 illustrates three components and how they are connected, all within the same composite.

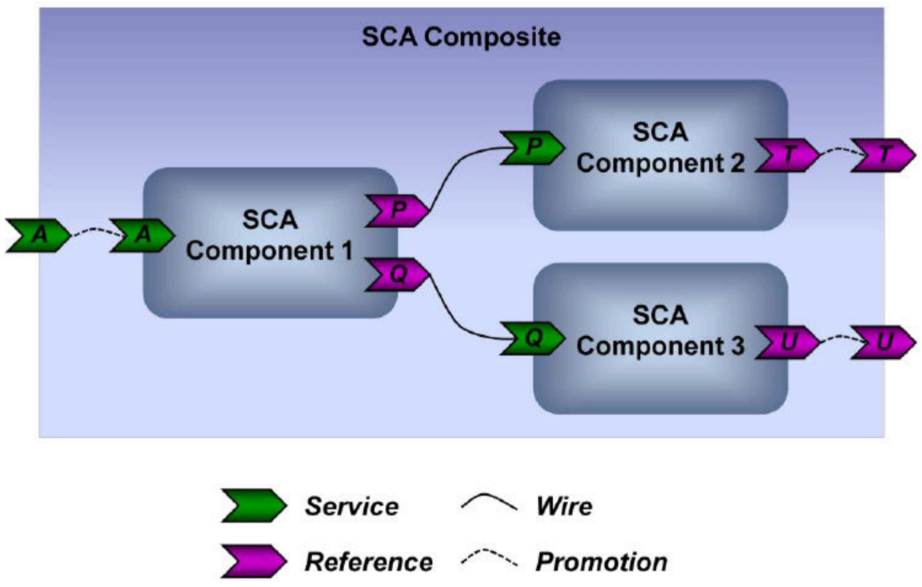


Figure 2.20: SCA composite example [36]

2.8.3.2 COMPONENTS

Components can be considered the atoms from which an SCA application is created. In other words, the components represents instances of implementation, that when compiled, provide the application its functionality. This functionalities can be implemented in object-oriented and procedural languages such as Java, C++, PHP or even XML centric languages such as BPEL and XSLT transformations [35]. There is no predefined list of supported languages, therefore, in theory, an SCA component can be implemented using pretty much any language or technology.

As Figure 2.21 depicts, an SCA component, it relies on a set of abstractions, including services, references and properties, in order to integrate with the SCA composite and specify its interactions.

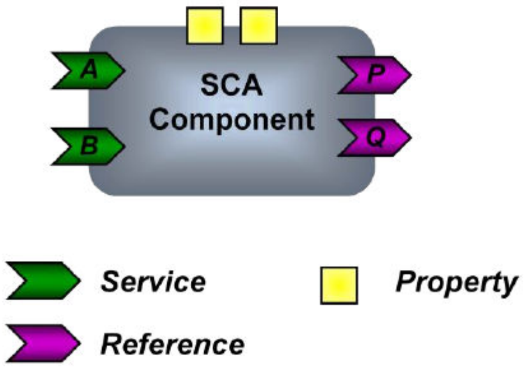


Figure 2.21: SCA component [36]

2.8.3.3 SERVICES, REFERENCES AND PROPERTIES

A service is the promotion of component functionality, that is, the exposed API of some or all implemented functionalities. Likewise, a reference represents a service from another component.

A property is a value typically defined in the SCDL configuration file, and it can be accessed from within a component. This type of approach enables for example the use of custom defined values per composite independently from the component implementation.

2.8.3.4 BINDINGS

The protocol or access mechanism in which the services and references can be accessed, is defined by the bindings. Furthermore, the services and references can have multiple bindings, for example, a service may have a SOAP binding and a REST binding in order to enable external access through web services of both types. This provides great flexibility by being independent from the component implementation.

Complex applications requiring multiple composites are expected on an SCA platform, but can introduce overhead depending on the protocol used to connect them. Therefore, to improve performance when connecting components from different composites running inside the same SCA runtime, an non-interoperable binding providing a highly optimized transport and protocol was created, called the SCA binding.

2.8.4 SCA IMPLEMENTATIONS

Despite being a reasonably new technology, there are already multiple solutions implementing it, some as adaptations to ESB platforms, as well as full flagged SCA solutions.

This section will describe some of the most relevant SCA solutions available today.

2.8.4.1 JBOSS SWITCHYARD

JBoss SwitchYard[14] is a component-based development framework focused on building structured, maintainable services and applications in a SOA environment. It was introduced in 2010 as the "Next-Generation ESB" at the JBoss WORLD Summit[37].

The project was build as a framework, running over the JBoss Enterprise Application Platform (EAP), this eases deployment and enables SwitchYard deployments to run alongside other Web applications (non-SwitchYard Web applications). The framework works with

[14]http://switchyard.jboss.org/

Apache Camel[15] providing a fast, simple and flexible integration runtime with its extensive connectivity and transports.

To facilitate and accelerate development, SwitchYard offers the SwitchYard Tooling, it acts as an IDE with visual feedback to help connect and configure the different elements when building SCA applications. The tooling is compatible with JBoss Developer Studio[16] and can also be installed as an extension to Eclipse IDE[17]. Figure 2.22 shows an application example in the SwitchYard Tooling.

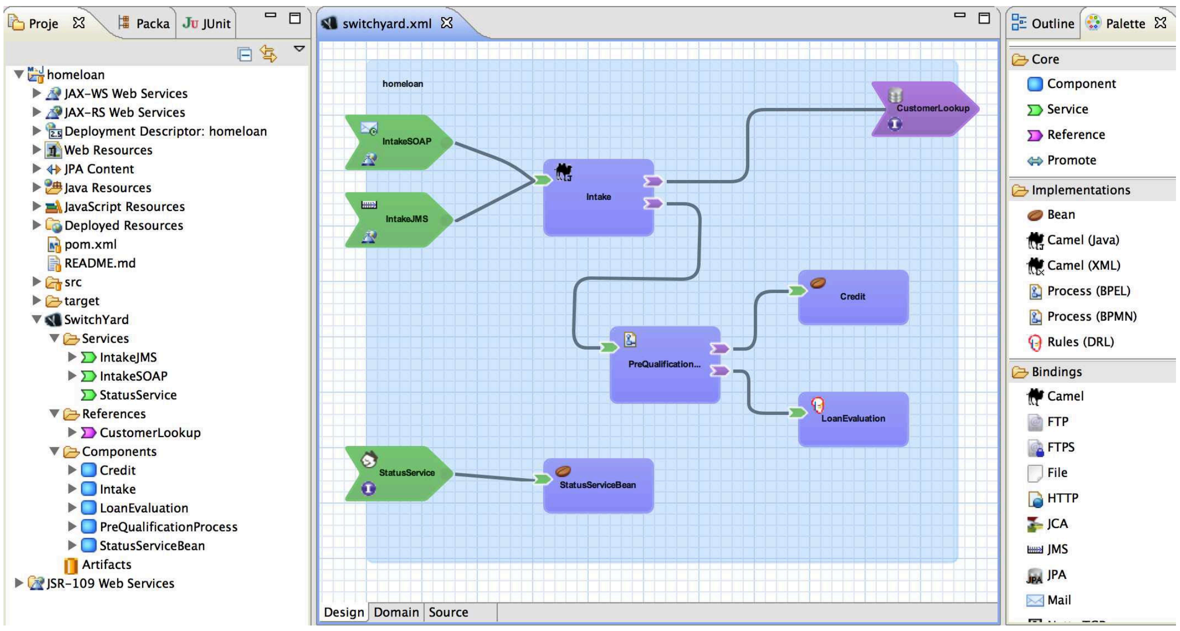


Figure 2.22: SwitchYard Tooling

One of the advantages of the SwitchYard framework, it's the vast number of supported technologies it offers. The following paragraphs will give an overview of the supported technologies per SCA element (for SwitchYard v1.1[18]).

Components. SwitchYard supports five different technologies to implement components, being they:

- **Java Beans:** Java EE class with dependency injection;
- **Apache Camel:** Implementation of Camel routes in Java or XML, leveraging the core routing engine inside of Apache Camel. It supports all the Enterprise Integration Patterns (EIP) from Apache Camel;

[15]Defined in Glossary
[16]http://www.jboss.org/products/devstudio/overview/
[17]https://www.eclipse.org/
[18]https://docs.jboss.org/author/display/SWITCHYARD11/Home

- **BPM:** Define business process with Business Process Model and Notation (BPMN);
- **Rules:** Implementation in business rules using Drools[19];
- **BPEL:** Implementation of the business process in Web Services Business Process Execution Language (WS-BPEL).

Bindings. SwitchYard offers an extensive list of supported bindings, the following list will give a brief description of each one:

- **SOAP:** SOAP-based web service;
- **HTTP:** HTTP binding, allow for example, a service to be invoked by a browser;
- **RESTEasy:** REST-based web service;
- **JCA:** Java EE Connector Architecture (JCA) binding, it allows to send and receive message to/from Enterprise Information Systems (EIS).
- **JMS:** Java Message Service (JMS), provides support for asynchronous communication with messaging providers;
- **File:** Provides filesystem level support;
- **FTP, FTPS and SFTP:** Provides support for remote file systems;
- **TCP and UDP:** Provides support for network level integration with TCP and UDP protocols;
- **JPA:** Provides support for operations regarding Java Persistence API (JPA);
- **SQL:** Structured Query Language (SQL) support for database operations;
- **Mail:** Support for consuming and sending mail messages;
- **Quartz:** Provides support for triggering services using *cron* expressions;
- **Timer:** Provides support for triggering services with fixed timer;
- **SEDA:** Provides asynchronous service binding between camel route and SwitchYard service;
- **Camel URI:** Allow Camel components to be used as gateway;
- **SCA** Binding used to connect SwitchYard services inside the same domain.

[19]http://www.drools.org/

2.8.4.2 FABRIC^3

Fabric^3 takes “a fresh approach to service-oriented applications”[20]. It is a platform to integrate loosely coupled systems and build distributed applications.

The platform is implemented in Java and can run as a standalone server, as well as to integrate with Application Servers as Apache Tomcat[21] and Oracle WebLogic[22].

Fabric^3 is architected as a small kernel, and therefore providing functionality through extensions. This type of approach can reduce the platform footprint, by choosing only the necessary extensions for the expected workflows.

As an SCA platform, Fabric^3 supports multiple bindings and types of component implementations. The next paragraphs will introduce those supported technologies.

Components. Fabric^3 supports three types of Java components, these components goes as follows:

- **Java:** Java class with dependency injection;
- **Timer:** Provides timer functionalities to Java classes;
- **Web:** Servlet and JavaServer Pages (JSP) support.

Bindings. Fabric^3 offers a number of communication mechanisms for performing remote service invocations and messaging. The following list will give a brief description of each one:

- **ZeroMQ:** High performance communication between clients and services in a publish subscribe environment;
- **JMS:** Use of Java Message Service (JMS) for message channels between client and services;
- **RESTFull:** Rest-based web service;
- **Web Services:** Web Services binding based in Metro[23];
- **File System:** File System support;
- **Web 2.0:** Supports exposing channels over websockets and long-polling;
- **FTP:** Provides support for remote file systems.

[20]http://www.fabric3.org/
[21]http://tomcat.apache.org/
[22]http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html
[23]https://metro.java.net/

CHAPTER 3

Architecture

The emergence of the IoT Platforms, allowed companies and users to connect their devices to the IoT with ease and low cost. Thus promoting the creation of contents without the concern of management and provisioning of their devices. To achieve that, there are some concerns that need to be addressed, such as: access control, M2M interoperability, service deployment and data analysis.

The IoT platform designed and implemented in this dissertation, was performed with a special focus on three points: 1) delegation of access control between multiple entities; 2) integration with ETSI standard; 3) dynamic service registration and deployment in a SOA platform.

This chapter aims to explain how those points were addressed and give a detailed overview of the project. The chapter is divided in three sections and each section represents a different level of abstraction, starting in the upper layer with the different entities and finishing with the inner structure of a service. The implementation details will be addressed in the next chapter.

3.1 First Layer (Entities)

The IoT Platform consists of several entities and some more as dependencies. This section is intended to provide a description of the entities present in a typical workflow as well to explain problems and solutions inherent to their interaction.

The Figure 3.1 represents the entities in a typical workflow, the following paragraphs will describe each one of those entities and their role with the IoT Platform in a service-oriented context. The last paragraph will address the challenges of Access Control in this scenario.

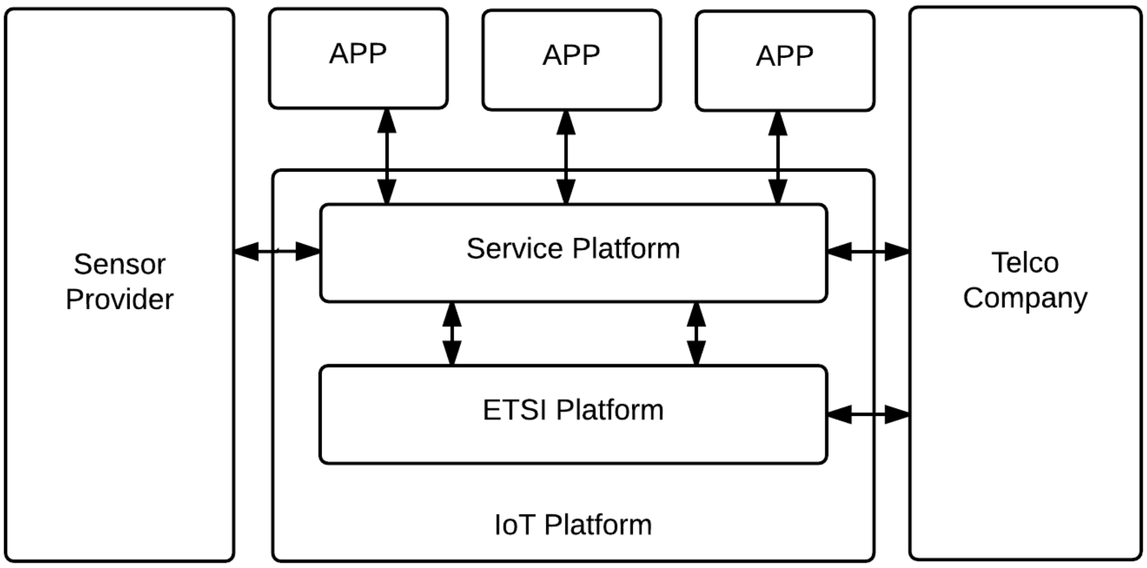


Figure 3.1: First Layer Architecture Block Diagram

IoT Platform. It's a complete IoT Platform, that joins device deployment and interoperability with service composition and deployment. This platform is composed by two different platforms, one to handle the devices and another to manage the services.

Service Platform. It's the service platform, all data to and from sensors is handled and routed here. It aggregates services in a SOA manner giving it a great extensibility and scalability.

ETSI Platform. It's the representation of an ETSI compliant platform.

Telco Entity. This dissertation was made in the context of an IoT Platform managed by a Telco provider. This is an expected scenario in the future.

Sensor Provider. This is the company who delegated their devices to the platform.

App. This is all and any app that can call services in the IoT Platform.

Access Control. One of the main focus of this dissertation was access control in an environment with multiple entities. When a user or company submits its devices to the IoT Platform, they need to maintain control over who can access their devices. As all data to and from devices is handled by services in a service oriented architecture, then also the access control will be better suited in this service layer. This also enables complex scenarios when delegation of authentication is needed. In this paragraph it will be assessed a complex use case with access control delegation in a four entities scenario.

Use Case. In this use case is described how can an App, known to the Sensor Provider but unknown to the IoT Platform access a service in the platform. The App has authorization from the Sensor Provider to access data from their services, but how can the IoT Platform recognize the App as a valid service consumer? In fact, the Platform not even recognize the Sensor Provider as trusted but only the Telco Company/Entity (Figure 3.2). How to solve this issue? The solution to this problem was solved creating a chain of trust, this can be a chain of digital signatures (Figure 3.3). As the Sensor Provider trusts the App and is trusted by the Telco company (made possible upon the devices delegation contract for example), and for the IoT Platform the Telco company is trusted, so as long as the request comes signed with a valid signature (signed by Telco) to the Platform it will be recognized as valid. Another problem that must be addressed is Authorization. How can a permissions based access be applied in this scenario? A company may have thousands or millions devices deployed, how to give access to just one of those devices and not all of them? This issue will be addressed in the next section 3.2.

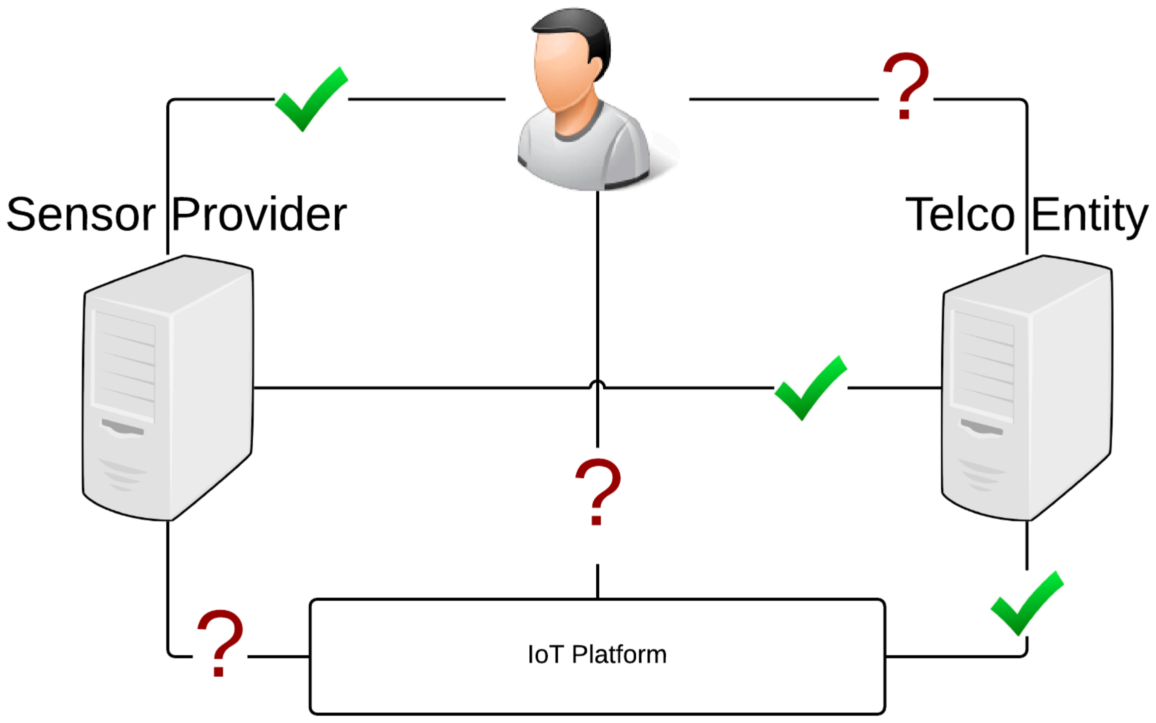


Figure 3.2: Diagram showing the trust connections between entities

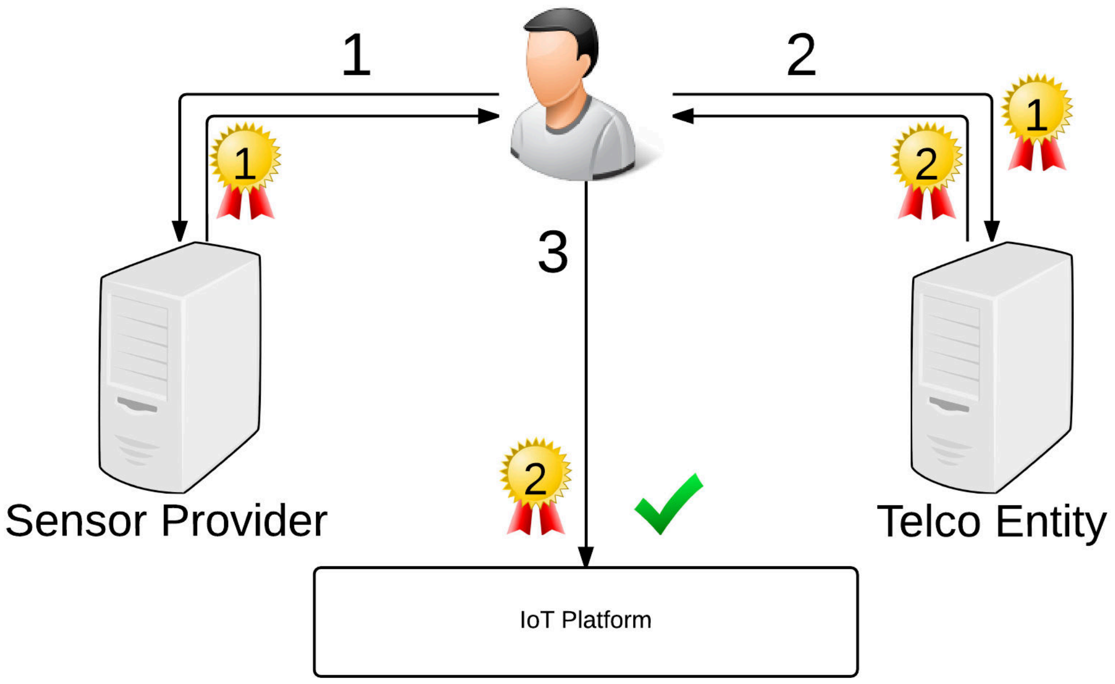


Figure 3.3: Diagram the chain of trust between entities using signatures

3.2 SECOND LAYER (SERVICES)

As a service-oriented platform some workflows may require multiple services. In this section and subsections it will be explained the most common services used and possible extensions.

Figure 3.4 represents the services needed for a typical workflow. The next subsections will explain the purpose of each portrayed service and give an overview explanation of the full workflow in the last subsection.

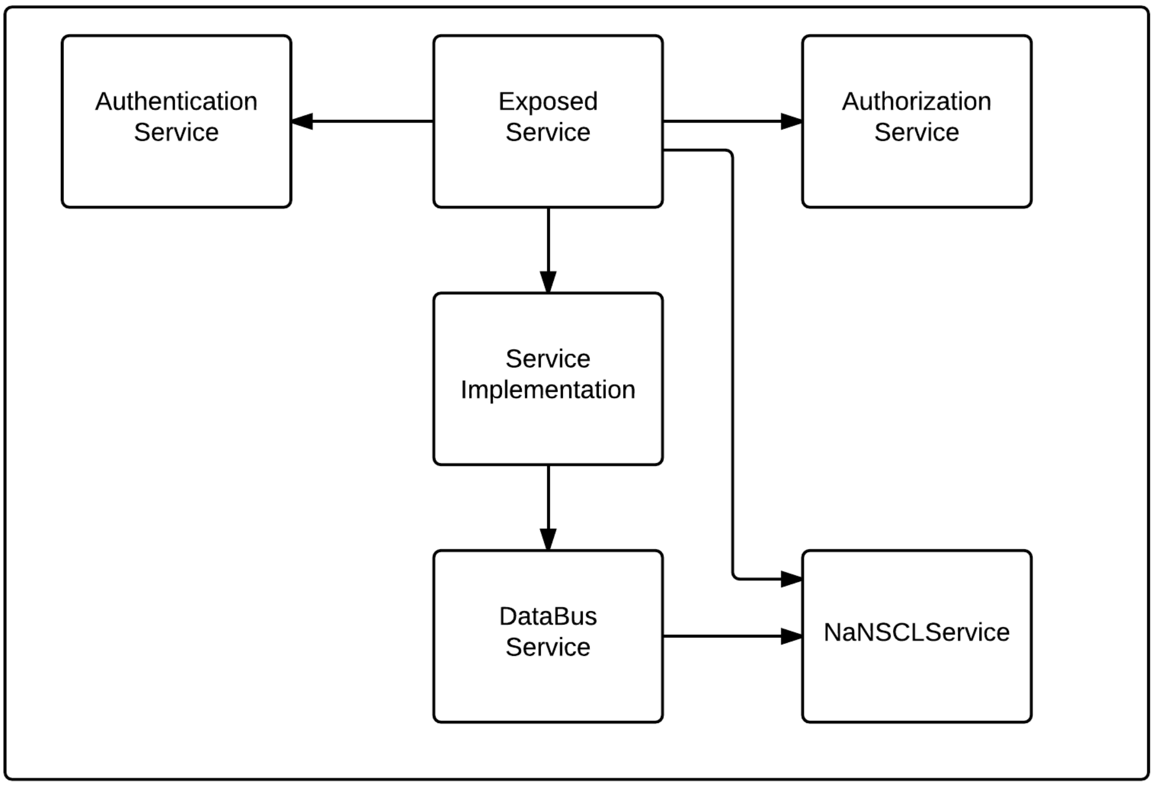


Figure 3.4: Second Layer Architecture Block Diagram (Generic Services)

3.2.1 AUTHENTICATION AND AUTHORIZATION SERVICES

The Authentication Service represents an implementation of an authentication mechanism or validation process. This eases the integration with other services and enables the creation of different authentication processes. Likewise, the Authorization Service performs the same functions but in an authorization context.

3.2.2 EXPOSED SERVICE AND IMPLEMENTATION

The Exposed Service will route through all security layers before calling the implementation, therefore securing the implementation with separation of concerns. Further explanations will be addressed in section 3.3.

The Service Implementation is the actual implementation of the service, it may be deployed locally or remotely, in case of a remote service a local service may act as proxy to facilitate integration. When data from sensors is needed, a connection with the DataBus (subsection 3.2.4) has to be established.

3.2.3 NANSCLSERVICE

This service is responsible for all direct interaction with the ETSI Platform, abstracting all the others services from the specifications of the M2M middleware. Therefore, it must support the following operations:

- Registration with the ETSI Platform to perform as a proper NA;
- Creation of containers and sub-containers;
- Post content in the containers and sub-containers;
- Get content instances from containers and sub-containers;
- Request subscription of containers and sub-containers.

Figure 3.4 presents two connections with the NaNSCLService, they designate two different use cases:

1. Resource subscription: this can be invoked in the initialization of a new service, thus enabling asynchronous scenarios and improving efficiency (this is the reason for the connection from the Exposed Service);
2. Get content instances: this represents a synchronous method to retrieve content from sensors.

3.2.4 DATABUS SERVICE

The DataBus, is a service to abstract all the data flow between services and the data coming from the ETSI Platform. Therefore, whenever a service within the Service Platform needs data from a sensor, it only needs to request it from the DataBus. Figure 3.5 illustrates the design of the DataBus, and as stated in that design with Services A, B and C, there aren't any specific differences between them, whereas they consume or publish information. This type of approach, makes a service more generic and grants greater flexibility, thus enabling new workflows, for example:

- Improve data flow performance using caches and queuing strategies;
- Data normalization;
- Statistical data analysis;
- Integration with Big Data services;
- Publish/Subscribe scenarios.

The Figure 3.5 also presents a connection with the NaNSCLService, this connection is needed to enable synchronous scenarios, like the request of information or data from a sensor in the ETSI Platform.

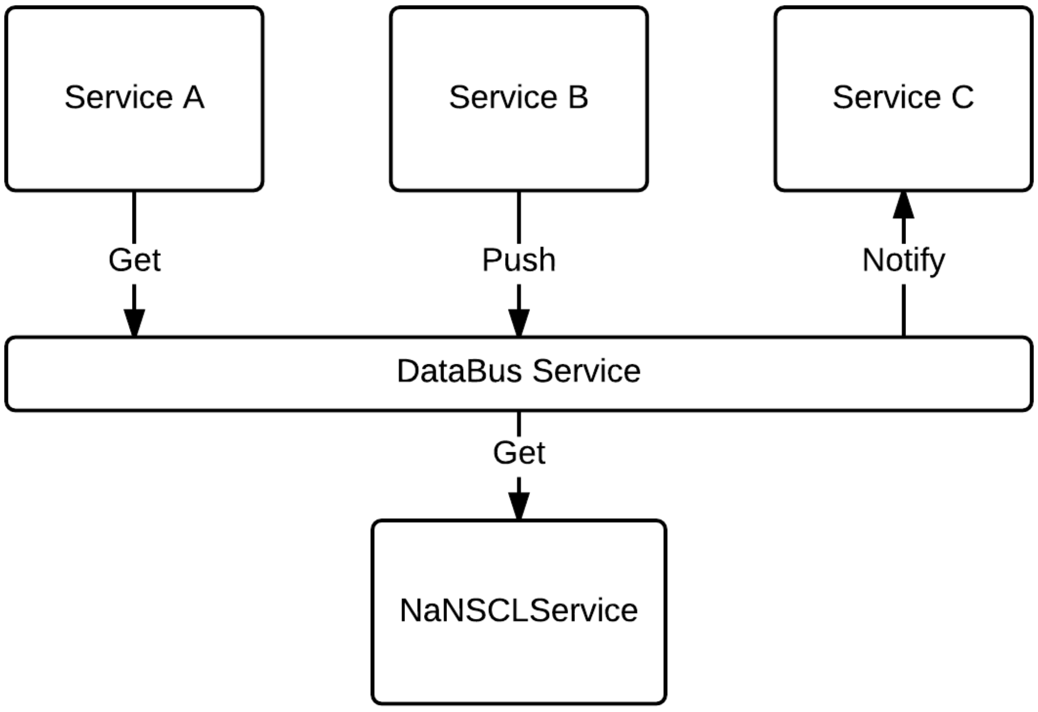


Figure 3.5: DataBus design

3.2.5 OVERVIEW

As described before, the Figure 3.4 presents the necessary services for a typical workflow. While previous subsections described the importance of each component, this subsection will explain step by step on how the workflow is performed.

The workflow can be divided in 6 different iterations, the following list will enumerate each iteration and explain what is performed:

1. **Service Bootstrap**: This happens upon the initialization of Exposed Service, a request to NaNSCLService is performed in order to subscribe resources that will later be necessary, although not mandatory, this can promote performance and enable asynchronous scenarios;

2. **Authentication**: An authentication process is performed to verify the clients identity, if it's not a valid client, an error will be thrown and the workflow will be stopped;
3. **Authorization**: Much like the Authentication process, in this step an analysis is perform to identify whether or not the client has enough permissions to invoke this service, stopping the workflow and throwing an error in case of insufficient permissions;
4. **Service Realization**: Now that all security mechanisms were performed and returned a positive response, the actual service implementation can be executed;
5. **DataBus**: In case the service needs to interact with data from any resource (sensors in the ETSI Platform) a request to the DataBus has to be performed;
6. **Get resource contents**: Finally, if gathering of new resource data is needed, a request to NaNSCLService is performed, retrieving that information from the ETSI Platform.

In conclusion, by dividing the business logic into several services, the extensibility is increased, enabling the addition of security layers and enhanced functionalities through service composition.

3.3 THIRD LAYER (SERVICE STRUCTURE)

In this section, will be performed a dissection on a typical service, presenting its inner structure and connections with other services. This service, already introduced as Exposed Service, is the service that can be reached from the outside and therefore the one responsible for security enforcement.

Figure 3.6 illustrates the inner structure of an Exposed Service and in gray the connections to and from this service. It introduces the External Client, which represents a client making a request to the Exposed Service, triggering the workflow presented in subsection 3.2.5. Inside the Service Platform, are also present four inner services (in gray), these services were already presented in previous subsections, therefore no further analysis will be performed.

The Exposed Service can be divided in 5 different components, the workflow generated by these components was already some how explained in section 3.2, hence, to prevent redundancy, only the External Endpoint will be address and no further explanations will be provided. The External Endpoint represents an endpoint visible from the outside, it's the endpoint that an external client can request.

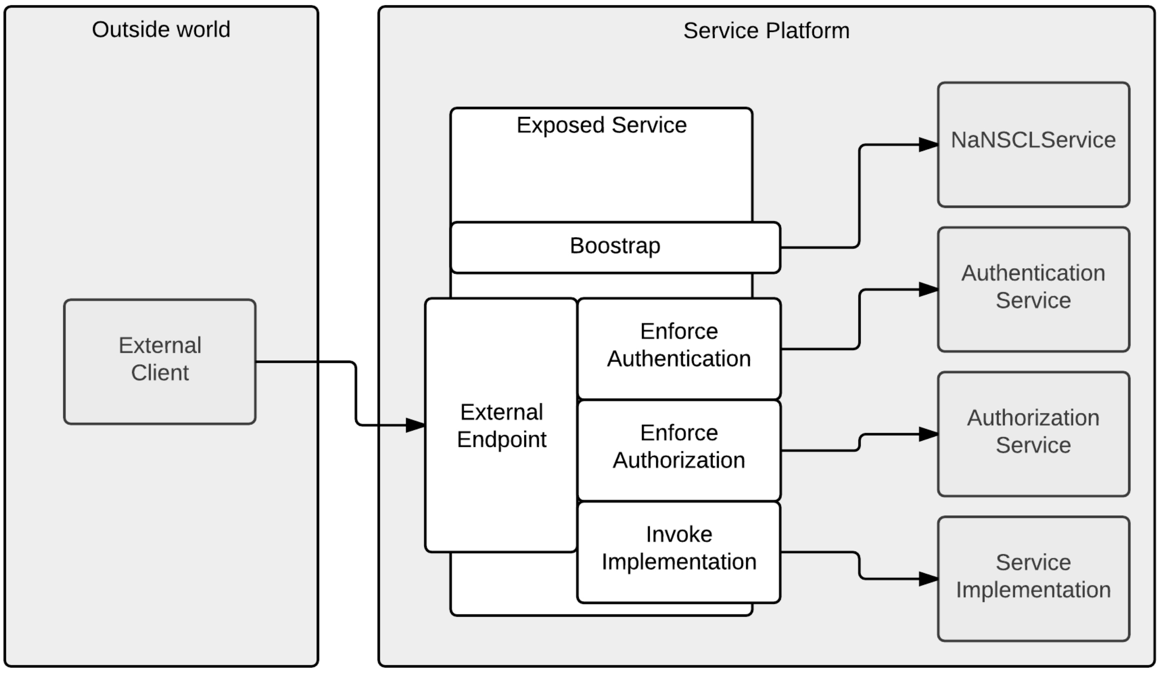


Figure 3.6: Third Layer Architecture Block Diagram

3.3.1 SERVICE ENCAPSULATION

In the scenario illustrated in the previous section, the service encapsulation it's of much importance. It's this type of design that will enable a fast service deployment, letting the developers/companies focus on implementation and be assured that security will be handled by the platform. Thus, also making possible a dynamic service registry, extending the platform and reducing the maintenance costs.

Another advantage in this type of abstraction, is the security it permits. By having a "shallow"/proxy service, enables the swap or the addition of new security layers without touching the implementation. While in the section 3.1 only authentication and authorization were needed, adding something like accounting could be done without much effort.

CHAPTER 4

Implementing an IoT Platform

This chapter aims to give a detailed explanation on how the implementation was done, presenting diagrams when appropriate and explaining the decisions made to implement and build this Service Platform for an IoT Platform solution.

4.1 DEFINED OBJECTIVES

This dissertation embraces multiple areas, ranging from federated security to service integration and data flow control. Despite that, four main objectives could be extracted from such a broad platform, being they:

- Analysis and possible solution for the multiple entity access control problem;
- Integration with an ETSI NSCL;
- Encapsulate service invocation (Mediation and Security enforcement);
- Routing sensor payload.

4.1.1 OVERVIEW

In the previous chapter were presented various high level entities in the context of the IoT Platform implemented in this dissertation. More specifically in section 3.1, where a description of these entities was made and their connections explained in a simplistic manner.

The Figure 4.1, shows a diagram equivalent to Figure 3.1 of that same section, but exchanging the theoretical names with the names of the software used to implement them.

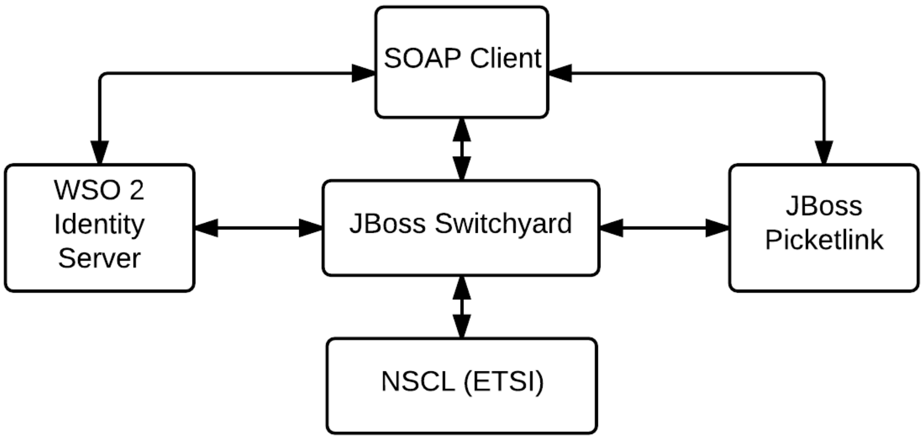


Figure 4.1: Entity Block Diagram

The next paragraphs will describe and explain the software solutions stated in the diagram and the reasons why they were chosen:

SOAP Client. This is not exactly a software platform but a roughly simple client, using SOAP as protocol for services requests. The client is not only simple because it needs to process multiple requests to different entities and build new messages (SOAP envelopes) depending on the responses it receives. This process will be explained in detail in the next sections.

WSO2 Identity Server. Although this entity is external to the IoT Platform, it could be seen as a partner because it is a crucial piece for the proper and possible functioning of the platform. As such, in order to design, implement and test afterwards, a solution was created to act as this entity. Thus, this entity besides entrusting sensors/devices to the IoT Platform it also needs to be able to authenticate its clients/partners/employees as well as to implement a granular permissions system, capable of handling complex authorization requests (i.e. a PDP). Hence, being a complete solution necessary to address authentication and authorization in this type of scenario, WSO2 IS came as a winning choice because of its easy deployment and powerful features. It was installed on a separate virtual machine to have a clear separation with the IoT Platform.

JBoss Picketlink. This entity has the function to sign and validate signatures, as well as to manage a user database. Although it seems an overly simplistic description of an IdP for a Telco company, in the context of this IoT Platform, it is just what is needed and thus abstracts from the inherent complexity of an entity of this type. As such, for test purposes, a simple service was created using JBoss Picketlink to handle signature and validation of x.509 certificates. The choice of this library, was just a practical decision, because the Service Platform is also a JBoss product and thus maintaining the same ecosystem would accelerate development and integration.

JBoss Switchyard. The choice of this solution turned out to be the most important technical decision throughout the dissertation. Since, it influenced the architecture of the whole system as well as its development. Its choice, required the consideration of several factors, but primarily because one of the objectives of this work was to have a platform of services in a SOA environment. By embarking on the SOA path, it falls instantly on the notion of ESB, and by studying existing solutions in this area, the SCA solutions appear as something new and even more steeped in the SOA essence then the ESB. As something relatively new, the idea of implementing a SCA Platform become enticing, and therefore the JBoss Switchyard was chosen as the platform to use. While it may not appear as stable as other solutions (fabric[3] for example), it offers a much wider range of software integration solutions, meaning more protocol bindings and more software integration technologies, concluding in a more flexible and extensible solution for an IoT Platform.

NSCL. This entity represents an ETSI compliant NSCL implementation, it will act as a middleware platform, acting as bridge between the Service Platform and the sensors/devices. It is a key component in the IoT Platform and it was developed in the context of another dissertation. Furthermore, the implementation of both platforms was done with a great level of communication between the peers, aiming a problem free integration.

4.2 ACCESS CONTROL

Access Control was one of the main objectives of this dissertation, as such, one of the most time consuming. This section, will give a detailed explanation of the implementation decisions and describe how everything fits together.

There are two main components in this access control: authentication and authorization. Each service with access from outside of the platform, has to ensure this two mechanisms. Therefore, and as explained in the previous chapter, a service template was designed to abstract the access control from the service implementation (Figure 4.2). Thus putting the enforcement of authentication and authorization at service level and not at platform level. Hence bringing a greater flexibility, allowing to add, change or remove security layers per service.

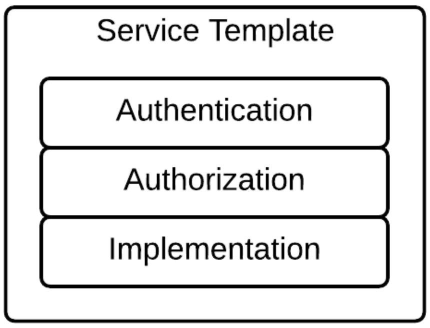


Figure 4.2: Service Template

4.2.1 THE CHAIN OF TRUST (AUTHENTICATION)

The authentication problem was already addressed in section 3.1 of the previous chapter and it is a well defined workflow between multiple entities. The objective of this subsection is to give a detailed explanation on how the entities interact as well as the processes triggered by that interaction. Figure 4.3 represents all the necessary entities for a correct authentication process. Comparing with Figure 3.1 from section 3.1, the new Figure presents the STS of the WSO2 IS. This STS will be responsible to authenticate the SOAP Client with the Sensor Provider as well as to generate a signed assertion, this will be later used by the SOAP Client to report to Telco Company.

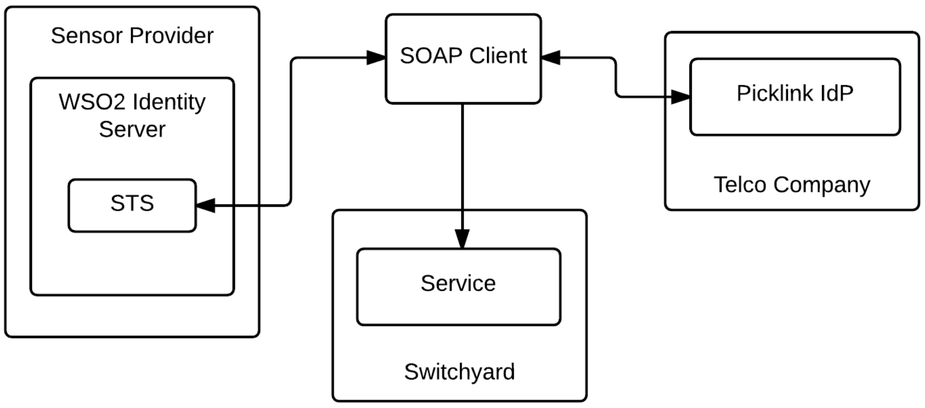


Figure 4.3: Authentication Entity Block Diagram

Figure 4.4 shows a sequence diagram in order to better understand how the message exchange is done and processed between the various entities until they reach the chain of trust needed to invoke the service.

The workflow starts when the SOAP Client authenticates itself with the Sensor Provider. This happens, because there is already a relationship between them and it is the sole responsibility of the two entities. Despite this process being external to the platform, for testing purposes, it was necessary to create this workflow to demonstrate the full process of authentication. In this particular case, it is used an STS to authenticate the client and

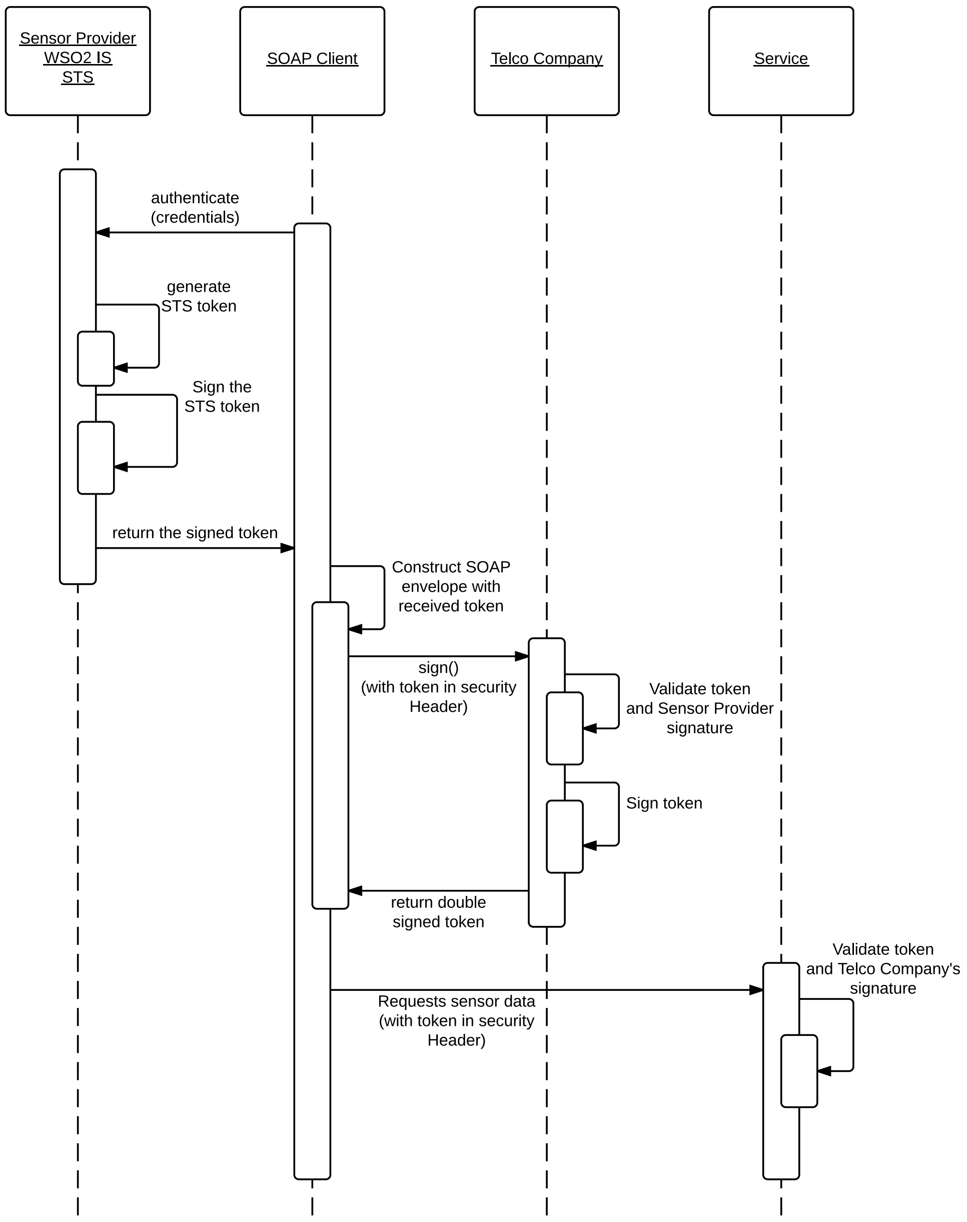

Sensor Provider WSO2 IS STS
SOAP Client
Telco Company
Service
authenticate (credentials)
generate STS token
Sign the STS token
return the signed token
Construct SOAP envelope with received token
sign() (with token in security Header)
Validate token and Sensor Provider signature
Sign token
return double signed token
Validate token and Telco Company's signature
Requests sensor data (with token in security Header)

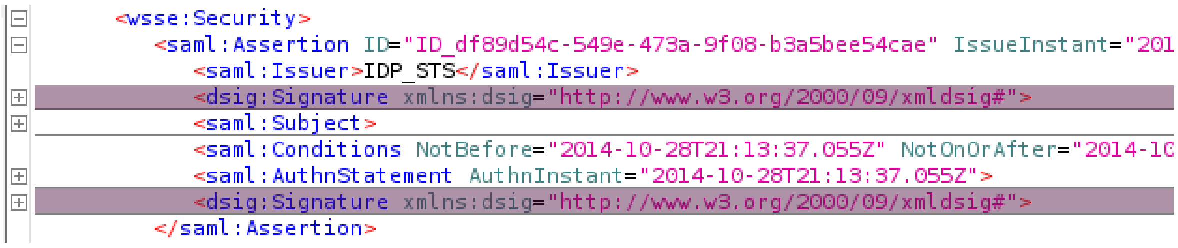
Figure 4.4: Authentication Sequence Diagram

generate a signed assertion, this assertion represents the client authentication as well as some data of this operation. Once generated and signed, it will be returned to the SOAP Client, and here ends the necessary interaction with the Sensor Provider in this authentication workflow.

Since it already has the signed assertion, the SOAP Client needs to insert it into a new SOAP envelope. The assertion (token and signature) is enclosed in the WS-Security (2.7.3) Header. Once assembled the envelope, the request for signature in the IdP Telco Company can be performed.

At Telco Company, upon receiving the sign (signature) request, a verification is performed to assure the existence of an assertion in the security header of the SOAP envelope. If true, it will scan all the signatures one by one (in case of multiple signatures, although in this workflow there is only one signature at the time) in order to find the signature of whoever issued the assertion. Upon finding the signature, if it was issued and signed by a trusted party, then the request is considered valid. Concluding the validation phase, the assertion will be signed in parallel with the other signature and returned back to SOAP Client. Here ends the role of Telco Company in this authentication workflow.

At this point the SOAP Client has the assertion with the two signatures (one from Sensor Provider and the other from Telco Company), an example of a double signed assertion can be seen in Figure 4.5. To invoke the Service, only the Telco Company signature is needed, but sending the double signed assertion will work as well, since it iterates through the signatures until a valid one is found. The invocation of the service represents the last interaction between entities in this workflow. That being said, once the signature is validated the Service is executed. Future requests to the Service may be made directly without having to perform the authentication workflow, provided that the signature and assertion are still valid. This facilitates and reduces significantly the request overhead.



```
<wsse:Security>
   <saml:Assertion ID="ID_df89d54c-549e-473a-9f08-b3a5bee54cae" IssueInstant="201
      <saml:Issuer>IDP_STS</saml:Issuer>
      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <saml:Subject>
      <saml:Conditions NotBefore="2014-10-28T21:13:37.055Z" NotOnOrAfter="2014-1
      <saml:AuthnStatement AuthnInstant="2014-10-28T21:13:37.055Z">
      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   </saml:Assertion>
```

Figure 4.5: Double Signed Assertion

4.2.2 AUTHORIZATION

An IoT Platform can deal with millions of connected devices, and each device can have different authorization policies. Thus, the platform must provide a fine grained authorization system to prevent unauthorized access to devices in both simple and complex scenarios. As explained in section 3.1, the devices managed by the platform are not owned by the platform, they are property of the Sensor Providers. Therefore, the point of decision for the authorization policies of each device must be handled with the Sensor Providers. When the subject who tries to access the devices is external to the platform (SOAP Client), the platform needs be able to translate that access request to a common language with the Sensor Provider for a correct evaluation. This type of scenarios could become very complex when the same service tries to access multiple devices with different policies. To meet these challenges, the XACML language appeared as the best choice, because it's extensible enough to achieve any type of authorization scenario (section 2.7.5).

This subsection will describe the authorization system implemented as well as to explain all the interactions between the different entities. Figure 4.6 represents the entities present in an authorization process. This diagram introduces two new actors:

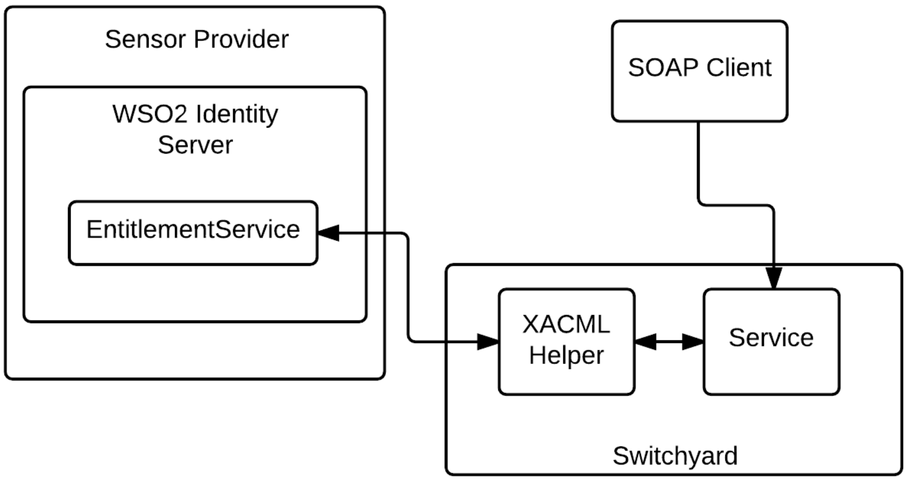


Figure 4.6: Authorization Entity Block Diagram

EntitlementService. The Entitlement Service is the XACML PDP functionality within the WSO2 IS. It is a full featured PDP, empowered by the WSO2 IS user interface to compile and edit XACML policies. Thus, as a solution used only for test purposes, all the "out of the box" functionalities eases the integration and test use cases.

XACML Helper. This is an inner service in the Service Platform. Hence, without any external endpoint, making it accessible only within the platform. Its only function, is to have a secure connection with the Sensor Provider PDP, therefore becoming the proxy between services and PDP.

Figure 4.7 represents the authorization sequence diagram. In that regard, a small description of each message transactions will be performed for a better understanding of the process.

Once the chain of trust is established, the SOAP Client is authenticated with the platform, although, further validation needs to be performed. Therefore, when a request to an exposed service is done, is assembled a complete report with all the necessary resources for the correct service realization. Hence, a policy query needs to be performed with the Sensor Provider PDP (EntitlementService), the XACML Helper will act as PEP and mediate the decision request with the PDP. Once a response is returned by the PDP, it will be handled back to Service. To finalize, if the response is "Permit" the service will proceed to the actual service implementation, else an error message will be return instead.

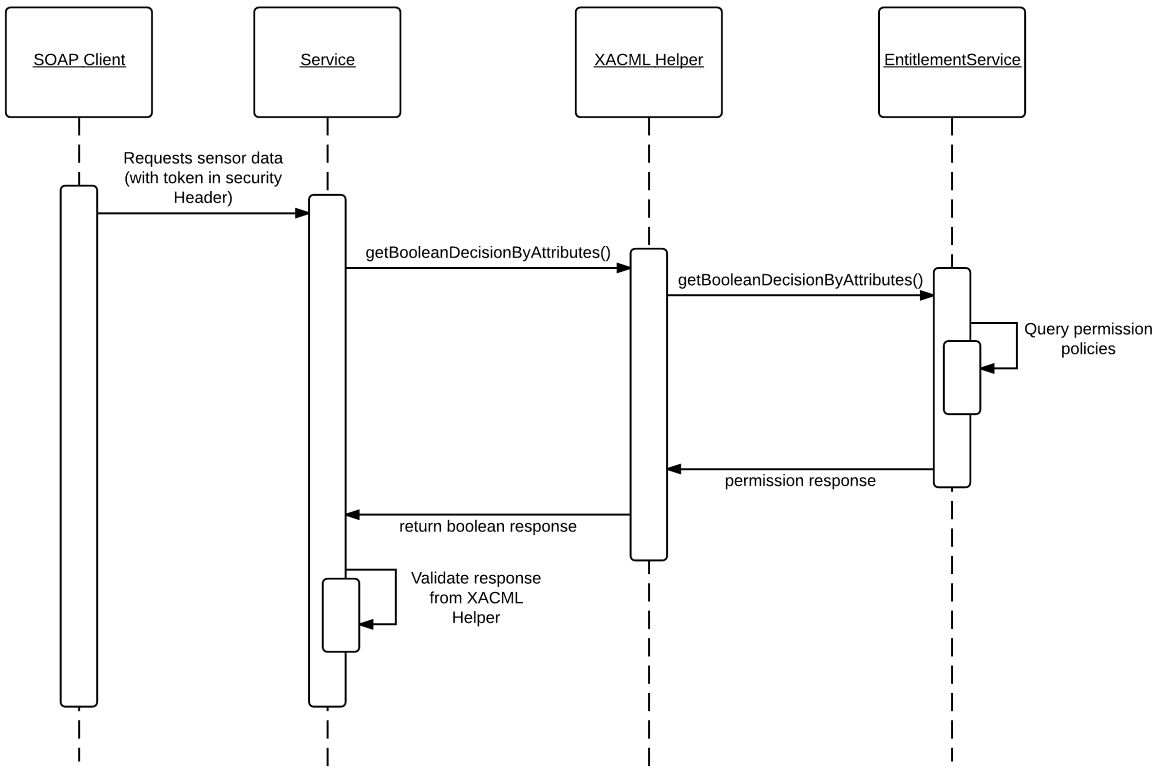

SOAP Client
Service
XACML Helper
EntitlementService
Requests sensor data (with token in security Header)
getBooleanDecisionByAttributes()
getBooleanDecisionByAttributes()
Query permission policies
permission response
return boolean response
Validate response from XACML Helper

Figure 4.7: Authorization Sequence Diagram

4.3 SERVICE MEDIATION

One of the objectives of this dissertation, was to ease service deployment by removing the security concerns from the developers, so they could focus only on implementation. Therefore, a separation of concerns between access control and service implementation was needed.

This section will explain how the separation of concerns was achieved by applying mediation mechanisms.

4.3.1 MESSAGE INTERCEPTOR

As explained in section 4.2, a requester, to be able to call a service in the platform, need to provide a valid assertion embedded in a WS-Security Header within a SOAP envelope. One way to prevent a malformed request, from reaching the service implementation, is to use a Message Interceptor. To accomplish that, the Message Composer from SwitchYard framework, was used. Therefore, upon arrival of a new request, the message is intercepted and checked to verify the existence of an assertion and if it is according to specifications. If it is not, a SOAP Fault is returned and the message doesn't reach the SCA Component with the service implementation.

Although this interception validate the existence or not of the assertion, it does not verifies its signatures. The reason behind it, is to separate the security specification from the authentication process, meaning, as it was previously specified, the request needs to be performed using SOAP with a WS-Trust valid assertion but the way it will be used for authentication may change or vary depending on the case. Hence, increasing flexibility and possible support for different authentication processes.

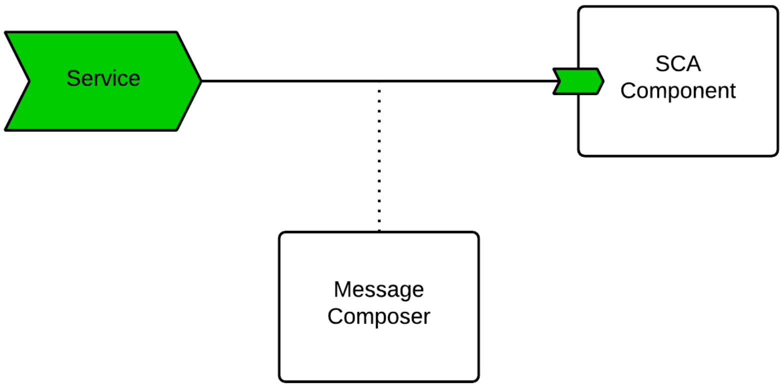


Figure 4.8: Message Interceptor (Message Composer)

4.3.2 SERVICE COMPOSITE

Service Composite is the component representation of a service in SCA (section 2.8.3), where protocols are binded and services and references are promoted. The Figure 4.9 illustrates a representation of this type for the Service Template introduced in section 3.3. The next paragraphs will give brief description on each component.

Service exposed and Service References. As a typical SCA service, it can promote an interface to a service endpoint, i.e. expose an endpoint internal or external in which the service responds when requested. The service endpoint represented in this diagram has as binding the SOAP protocol (it was explained in section 4.2 the reason why SOAP was chosen as protocol). This endpoint is the green component in the Figure.

Concerning the Service References in the Figure (represented by the purple components), they are inner connection to other services within the Service Platform (Switchyard). The references use as binding the SCA protocol which facilitate inter-application communication within a SwitchYard runtime (section 2.8.3). This service was already introduced in previous section, thus for simple reminder, a small list follows as:

- **SigningService:** service to validate signatures;
- **XACMLHelper:** service to perform XACML request to a PDP and therefore act as PEP;

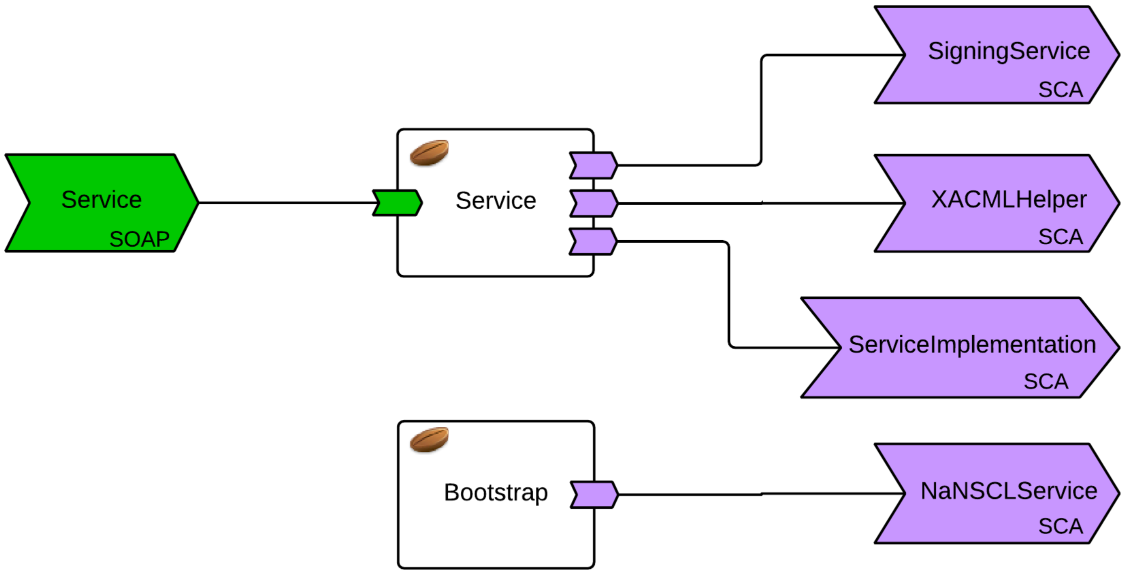


Figure 4.9: Service Composite

- **ServiceImplementation:** actual implementation of the service;
- **NaNSCLService:** service responsible for every interaction with the NSCL.

Bootstrap. This is not a mandatory component, but it serves as a requisite disclaimer. In other words, it maintains a list with all the necessary resources (sensors/devices) for a correct service realization. Thus, to enhance performance or to support asynchronous data flow, resource subscriptions can be performed to the NSCL having NaNSCLService as intermediary. Use cases of asynchronous data flow between the Service and ETSI Platforms is addressed in section 4.5.3.

Service. This is the component where the business logic of the Service Template is specified. It is where the security layers will perform their duty, and the routing between references is coordinated. This component is a Java Bean, and therefore, implemented in Java. While a business process language (BPMN or BPEL) could be used, as it is supported by the SwitchYard framework, the programmatic approach turned out to be more flexible and easy for implementation and debug.

The Figure 4.10 illustrates the business logic of the Service Template. The first validation comes with the signature verification, the assertion is extracted from the SOAP Envelope (which is delivered by the Camel context from the request) and forward to the referenced service for signature validation. Upon success, a second validation will be performed, this time concerning permissions (authorization). Therefore, the assigned service for authorization will be invoked, in order to verify if the user(requester) as enough permissions to read or write on all the resources needed for a complete service realization. If both security layers (authentication and authorization) return "true", then the real service implementation that is assigned as reference, can be invoked. Although, if any of the verifications return "false", an exception will be returned to the requester.

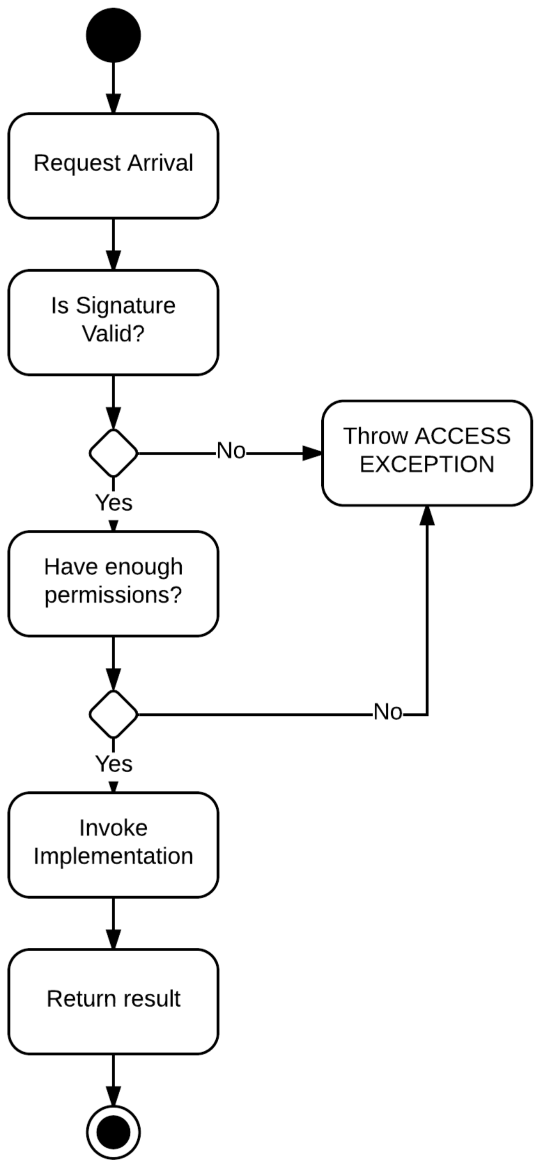


Figure 4.10: Enforcement Activity Diagram

4.4 ETSI INTEGRATION

In order to create an IoT Platform, the Service Platform is not enough, it is also necessary a middleware platform to handle all M2M communications as well as the access to that information. Hence, as already explained, an integration with an ETSI compliant NSCL was performed.

This section aims to describe the interactions with the NSCL and how they were accomplished.

4.4.1 CONNECTIONS

ETSI specifies a wide number of restful APIs for the communication with the SCLs. Therefore, all synchronous communication between both platforms (services and NSCL) are processed over those restful APIs.

To handle this interactions, a service by the name of NaNSCLService was created. This service is responsible for all communications with the NSCL, and serves as proxy for other services who require communication with that platform. The supported procedures goes as follows:

- **NA registration:** This will register an entity and return an id. Now the registered entity can perform other requests and is identified within the ETSI platform, which in turn will allow the permissions system in the ETSI to control the access of that entity, to the resource containers.
- **createContainer:** Request for the NSCL to create a container.
- **createSubContainer:** Request for the NSCL to create a sub-container.
- **getContainerInstances:** Retrieve the contents from a container or sub-container.
- **postContainerInstances:** Post contents to a container or sub-container.
- **getLatestContainerInstances:** Retrieve the latest contents from a container or sub-container.
- **postSubscribeContentInstances:** Subscribe to a container or sub-container, whenever a new content is available, it will be pushed to a message queue.

Every request and response to and from the NSCL, must be encompassed with XML-based message within the body of the request/response. For different methods, different messages are needed, and to ease the interpretation as well as the assembly of those messages, a SDK was provided by the developer of the NSCL implementation.

Note: as specified in the ETSI standard, in case of problem with the request or in its process, the response must contain an HTTP Status Code of error as well as an XML message in the body referencing the problem. In SwitchYard 1.1, the library responsible to process the REST requests and responses is the RESTEasy JAX-RS(version 2.3.6.Final-redhat-1), however this library when receiving a REST response with an HTTP Error Code, throws an inner exception and discards the response. This prevents error handling at service level (in this case it would be the NaNSCLService). As a workaround, a small change was performed on that library, to not discard those types of messages, making error handling possible at service level.

4.5 DATABUS

In order to mediate, facilitate and improve the access of sensor data to services in the platform, a service by the name of DataBus was created (it was first introduced in section 3.2.4).

This section will explain the possible use cases as well as some of the particularities and features from the DataBus implementation.

4.5.1 OVERVIEW

The DataBus offers two types of methods: Get methods and Post methods. As seen in Figure 4.11, the Get methods will return sensor data while the Post methods will push data to the Service. By pushing data, it allows to cache that same data as well as to enable publish and subscribe scenarios. It also provides, the ability to connect other synchronous and asynchronous data sources, as the insertion of data in the DataBus is not binded to any specific service. An example, could be an helper service connected to a different M2M middleware, pushing new data to the DataBus.

Concerning security, the DataBus service only have SCA bindings in its interfaces, this prevents direct access from the outside as the SCA binding only accepts connections from within the same SwitchYard domain.

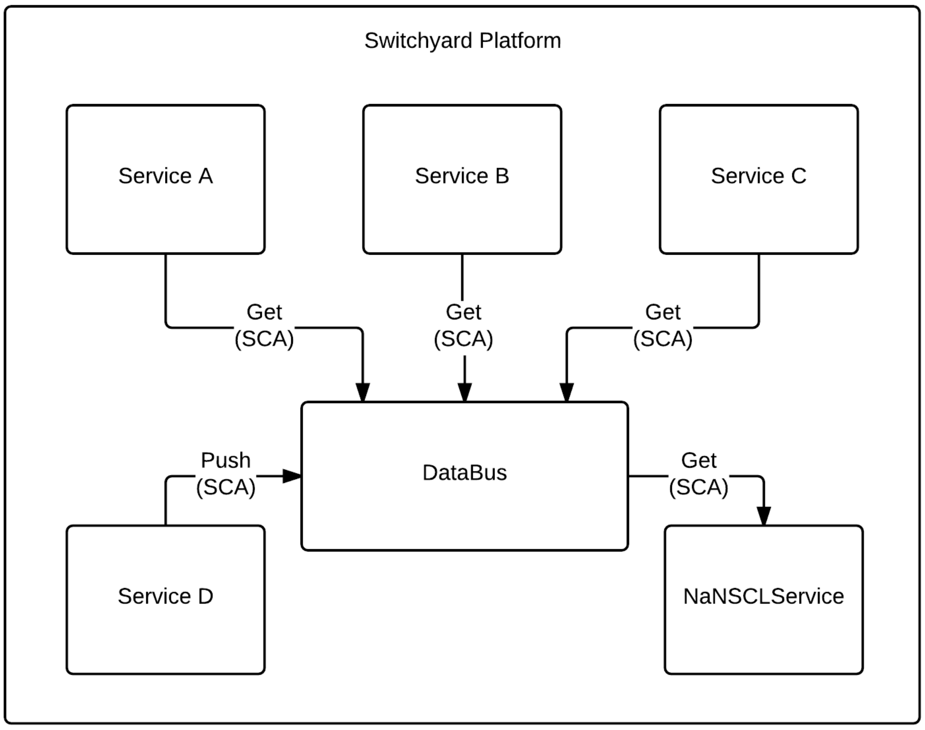


Figure 4.11: DataBus Overview

4.5.2 CACHE SYSTEM

The DataBus service, being the only source of data within the Service Platform, can have an overwhelming number of requests. That being said, there is good chance, that more than one service wants the same data from the same resource. Therefore, in order to improve performance and avoid duplicate requests to the NSCL (through the NaNSCLService), a cache system was implemented.

The cache system implemented, uses a Redis[1] database. The reasons to choose Redis as database, derives from its easy store and read methods, its fast performance and the simplicity it brings to the implementation. Moreover, the cache system wasn't one of the objectives, it was simply a *nice to have* feature, therefore a simple implementation would suffice.

4.5.3 USE CASES

The DataBus service, supports multiple use cases due to its flexible design. This section will present the use cases implemented in this dissertation.

[1]Defined in Glossary

4.5.3.1 SYNCHRONOUS SERVICE DATABUS INTERACTION

The Synchronous Service DataBus interaction is the simplest use case described in this section. It takes place when a service requests data from a specific resource to the DataBus. If the data from the requested resource is available at the cache system, it will be returned right away to the requesting service. If not, the use case in subsection 4.5.3.2 takes action, and once finished, the result is returned. This use case is illustrated in the Figure 4.12.

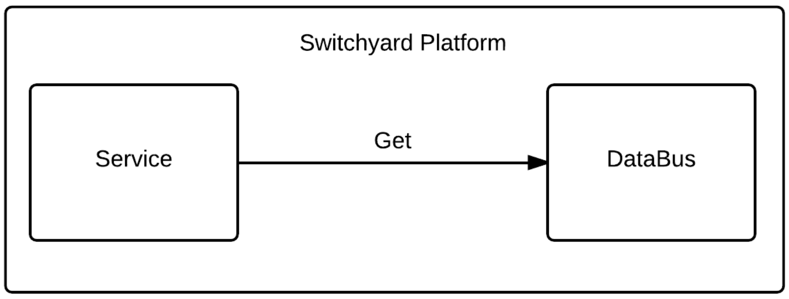


Figure 4.12: Synchronous Service DataBus interaction

4.5.3.2 SYNCHRONOUS CONTENT INSTANCES REQUEST

When in need for new resource content. A request to the NaNSCLService has to be performed, as it will act as "proxy" between the DataBus and the NSCL. This request will ask for content instances of a specified resource. Finally, if the request with the NSCL returns valid content, it will be relayed back to the DataBus. This use case is illustrated in the Figure 4.13.

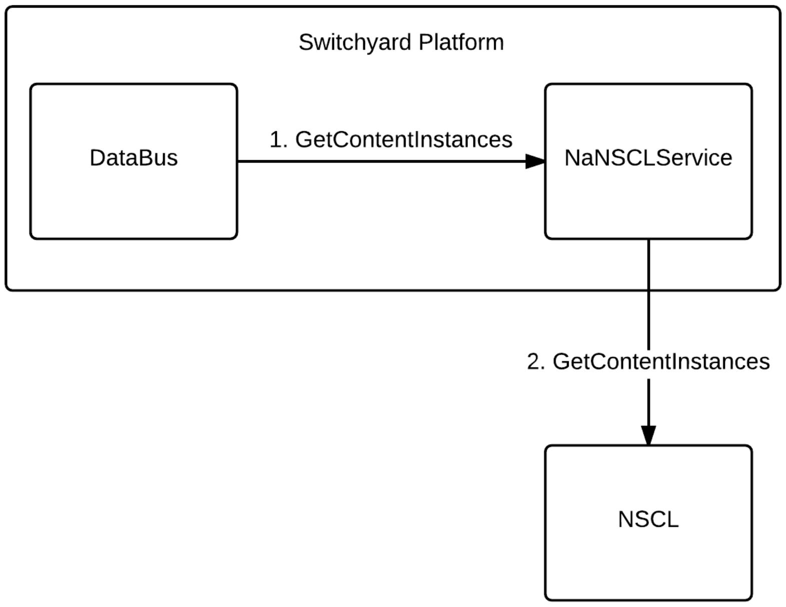


Figure 4.13: Synchronous content instances request

4.5.3.3 ASYNCHRONOUS UPDATES FROM NEW CONTENT INSTANCES

This use case is the most complex case described in this section. It represents the process of notification when new data is made available by a resource. The Figure 4.14 represents this use case and introduces a new service, the ActiveMQ Helper. This new service will handle all the publish subscribe pattern, and by doing so, abstracts this process completely from the DataBus. As the name implies, this helper service is connected with an ActiveMQ queue, and the reason for choosing the ActiveMQ as message broker is external to this dissertation, since it was responsibility from the NSCL developer.

For an improved understanding of the diagram illustrated in the Figure 4.14, a numbered list representing each interaction will be presented below:

1. **Subscribe Resource:** It's the process of subscribing a specific container, this container represents the data produced by a resource. This process could be initiated by the initialization of a Service Template, which inform the NaNSCLService of the resources it requires.
2. **Update container:** It represents the creation of new data by the resource.
3. **Publish:** The updated container generates a publish process, which will send the new data to a previous defined message queue.
4. **Consumer notify:** Whenever the ActiveMQ receives new content, it will notify all consumers and deliver to them the new content.

5. **Push data:** Upon notification, it routes the received data to the DataBus Service.
6. **Store data:** For caching purposes, the new data will be stored in a Redis database.

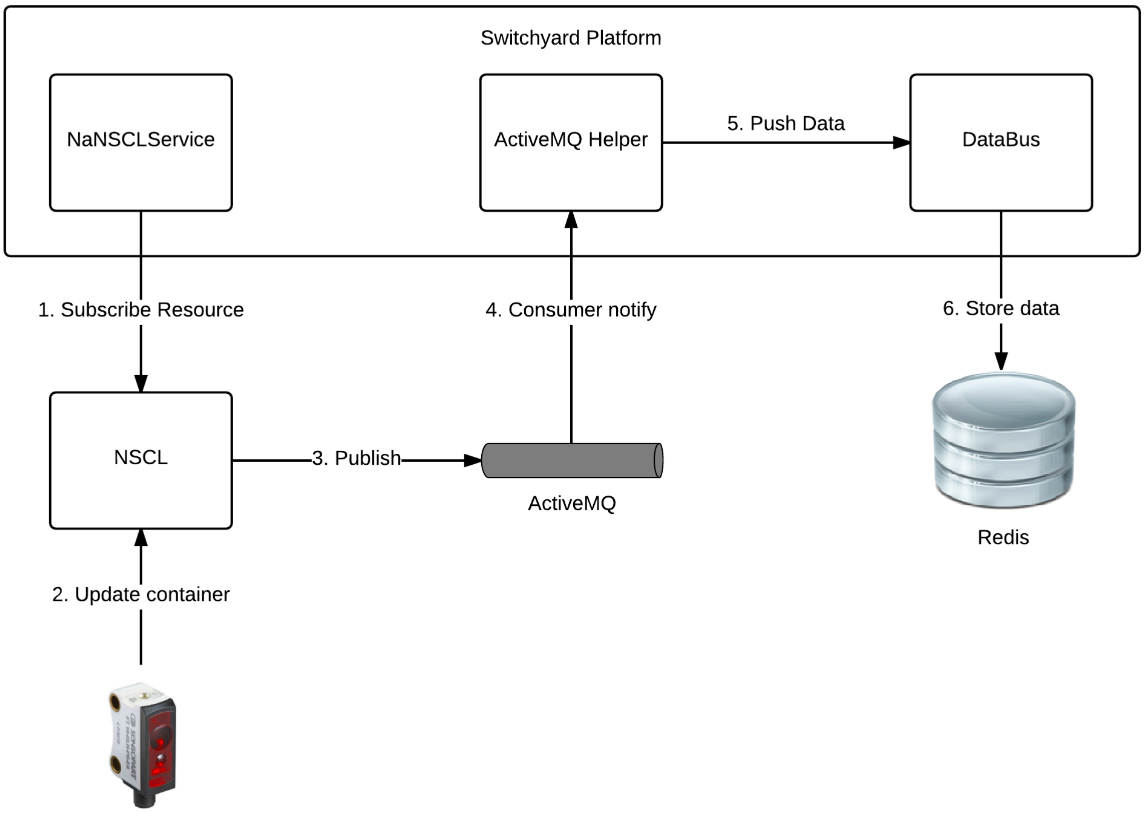


Figure 4.14: Asynchronous updates from new content instances

CHAPTER 5

Evaluation and Results

As a Service Platform for a IoT Platform, performance is of much importance, since it may handle a huge number of devices and service requests. This dissertation had a great focus on security, more precisely, access control and it is undoubtedly that security measures always have impact on performance.

This chapter will test the implemented platform in terms of performance, with special focus on the impact that security creates.

5.1 Deployment Scenario

The Figure 5.1 illustrates the test case scenario assembled to analyze and study the platform in terms of performance. It is composed by two types of components: Machines and SOAP Clients.

The Machines are the representation of the physical and virtual computers, used to deploy and support the multiple solutions needed for a complete use case. The Table 5.1 presents the specifications of this computers. Concerning the virtual machines, they were running over VirtualBox 4.3.14.

There are two different Clients: one in Java and the other using SoapUI. The Java Client is used to initiate the Authentication process in order to retrieve the double signed assertion (as explained in section 4.2.1). The second Client uses the Test Suites features from SoapUI, which enables load tests for simulating multiple clients requesting at the same time.

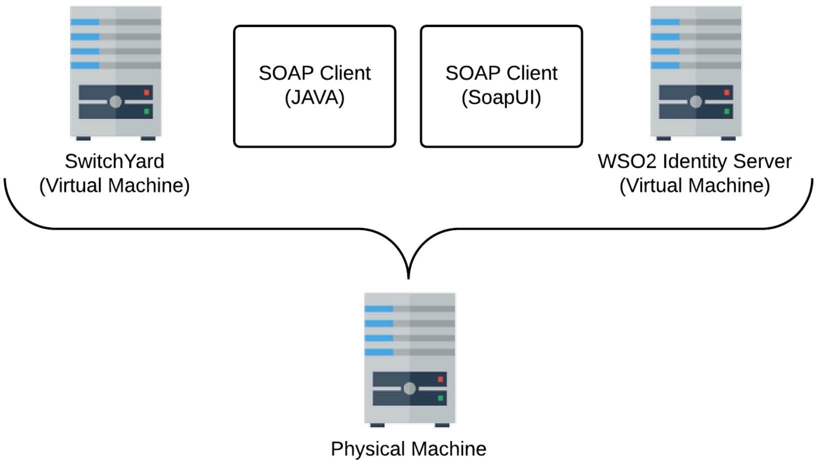


Figure 5.1: Deployment Scenario

Machine	Operating System	CPU	RAM
Physical Machine	Arch Linux on Linux kernel 3.17.1 (64-bit)	Intel(R) Core(TM) i7-2630QM @ 2.0-2.9 GHz	8 GB @ 1333 MHz
SwitchYard	Ubuntu Desktop 12.04 (32-bit)	1 Core CPU (with accelerated virtualization)	2 GB
WSO2 Identity Server	Ubuntu Server 12.04 (32-bit)	1 Core CPU (with accelerated virtualization)	2 GB

Table 5.1: Machines specification

5.2 TEST CASE

Figure 5.2 represents the test case scenario. The workflow objective, is to request the system to retrieve one value from the DataBus that is stored Redis Database and return it to the original requester (SOAP Client).

This test case aims to determine the performance costs created by the security stack. It will measure time and throughput of multiple requests happening at the same time, in a 60 seconds period.

The tests were performed with 5 threads, requesting the same service, consecutively, during 60 seconds. There were 5 different scenarios:

1. **Request to Mediator**, this represents the typical workflow, in which full mediation is performed - authentication, authorization, implementation;
2. **Request to Mediator without authentication**, in this case, signature validation is not performed;
3. **Request to Mediator without authorization**, in this case, authorization is not performed, therefore, no connection to WSO2 Identity Server Machine is necessary;

4. **Request to Mediator without security**, in this case, it is verified the existence of an assertion, but no further validation is performed;
5. **Request directly to Service Implementation**, without performing any kind of mediation or security validation.

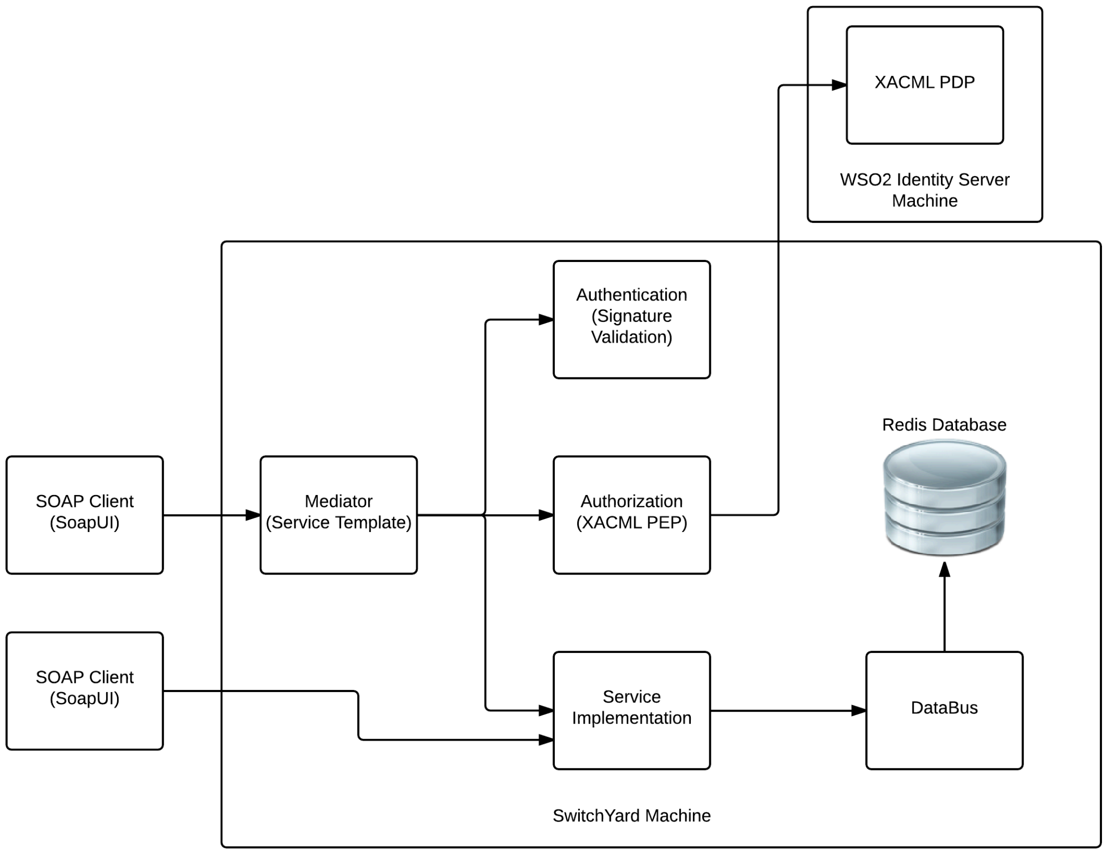

XACML PDP
WSO2 Identity Server Machine
Authentication (Signature Validation)
Redis Database
SOAP Client (SoapUI)
Mediator (Service Template)
Authorization (XACML PEP)
SOAP Client (SoapUI)
Service Implementation
DataBus
SwitchYard Machine

Figure 5.2: Test Case Scenario

5.3 RESULTS

The values obtained for the scenarios enumerated in the previous section, are discriminated in Table 5.2. For a better understanding, the following list explains the meaning of each column:

- **min**, the shortest time of a request (in milliseconds);
- **max**, the longest time of a request (in milliseconds);

- **avg**, the average time of the requests (in milliseconds);
- **last**, the last time of a request (in milliseconds);
- **cnt**, the total number of requests;
- **tps**, the number of transactions per second;
- **bytes**, the number of bytes processed by the requests;
- **bps**, the number of bytes per second;
- **err**, the number of errors;
- **rat**, failed requests ration.

Test Case	**min**	**max**	**avg**	**last**	**cnt**	**tps**	**bytes**	**bps**	**err**	**rat**
Mediation	38	236	55.83	85	371	6.15	115752	1921	0	0
Mediation no Authentication	22	163	30.71	30	380	6.32	118560	1972	0	0
Mediation no Authorization	28	111	42.08	64	384	6.35	119808	1982	0	0
Mediation no Security	12	151	16.98	12	390	6.49	121680	2025	0	0
No Mediation	9	42	11.9	10	389	6.46	135761	2257	0	0

Table 5.2: Test results

Analyzing the results in terms of average request time, a notable difference between scenarios can be observed (Figure 5.3 helps to visualize those discrepancies). In that regard, the Table 5.3 was created, were it states clearly the cost of each component, and by summing the average differences, comes a value very close to the one obtained in the first scenario ($55.83 \approx 55.81$).

It is clear, that the most costly component, is the authentication. Comparing with authorization, the latter is much less heavy on performance, despite the necessary request to an outside Virtual Machine (WSO2 Identity Server). The possible reasons for that, could be that the authentication process handles XML elements (which can be costly to performance) as well as the validation of an X.509 certificate signature.

Test Case	**average**
No Mediation	11.9
Mediation no Security	+5.08
Authorization	+13.73
Authentication	+25.1
Total	**55.81**

Table 5.3: Performance cost per component

Regarding throughput, it is clear in the results (Table 5.2) that the number of bytes processed and per second decreases with the addition of security, but by comparing the values,

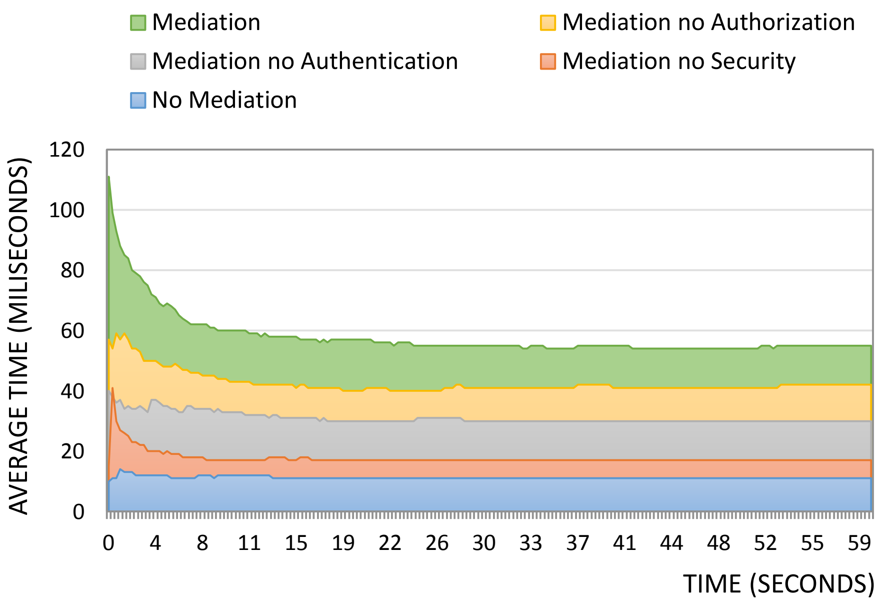


Figure 5.3: Average request times per scenario

of the worst (115752 bytes) and the best (135761 bytes) values in bytes processed, they only have a difference of 1.173 % (135761/115752). Besides, the total number of requests as well as transactions per second, also don't have big impact in the various test cases.

To conclude, the most heavily impacted metrics between the different cases, are the request times. Therefore, in order to improve performance, optimizations could be done in the security mechanisms. The flexible design of the platform, also makes it possible to add different security mechanisms, for example, enable simplified security for uses case where security is not a priority.

CHAPTER 6

Conclusion

The recent demand for IoT solutions and the exponential growth of M2M connections has led to the creation of many IoT Platforms, these solutions focus on content, as they provide easy integration of devices and application deployment, and consequently bringing the world closer to a global IoT scenario. This document provides an overview on some of the most important projects existing today as well as different approaches used to assemble an IoT Platform.

However, the main goal of this dissertation was to create a Service Platform integrated with an M2M middleware in order to achieve a complete IoT Platform. In that regard, interaction with a working ETSI M2M middleware through an NA was accomplished, creating a complete set of workflows to handle and interpret sensor data.

The platform was designed and implemented with separation of concerns in mind, clearly separating access control from service implementation and abstracting data access from the middleware specification. Therefore, enforcing authentication and authorization on independent service implementations through mediation. The platform was tested to demonstrate functionality as well as performance and performance costs for the multi layer access control and mediation.

Finally, the architecture design, allow great extensibility by abstracting the data flow from the service, creating a content focused development where security and data access is automatically binded to the implementation effortlessly.

6.1 FUTURE WORK

Although the provided implementation is working and fulfilled all the proposed objectives, there is still room for improvement and *nice to have* features, that would elevate the platform to a more complete and competitive IoT Platform for todays market. The following list describes some of those possible improvements:

- **Optimize Signing service:** The tests presented in chapter 5, reveal that the signature validation is the most expensive component in the security stack, therefore optimization is necessary;
- **New workflows for DataBus**: A publish subscribe workflow between DataBus and platform services would improve greatly asynchronous scenarios, allowing service realization upon the arrival of new sensors data;
- **Service deployment system**: The separation of concerns between service implementation and access control, enables developers to focus on content creation (service creation), and a framework, platform or user interface to enable developers to deploy their own implementations as well as define service dependencies and security systems in an easy to do configuration, would allow rapid growth to the platform.

Glossary

Apache Camel	Apache Camel[1] is open source integration framework based on Enterprise Integration Patterns, it provides mediation and routing rules through configuration in a variety of domain-specific languages.
Identity Provider	IdP is an entity responsible for issuing identification information.
Redis	Often referred to as data structure server, Redis[2] is an open source, networked, key-value cache and store.
Security Token Service	STS is a service component, normally a software based IdP, that issues security tokens, as part of a claims-based identity system.
Service Provider	SP is an entity that provides a service.
Single sign-on	SSO is an authentication process to allow users to authenticate once, and therefore be logged into multiple systems without further manual intervention.

[1]http://camel.apache.org/
[2]http://redis.io/

References

[1] J. Holler, V. Tsiatsis, C. Mulligan, S. Avesand, S. Karnouskos, and D. Boyle, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. Elsevier Science, 2014, pp. 9–39, ISBN: 9780080994017. [Online]. Available: http://www.google.pt/books?id=wtfEAgAAQBAJ.

[2] A. Qureshi, W. P. Kang, J. L. Davidson, and Y. Gurbuz, "Review on carbon-derived, solid-state, micro and nano sensors for electrochemical sensing applications", *Diamond and Related Materials*, vol. 18, no. 12, pp. 1401–1420, Dec. 2009, ISSN: 09259635. DOI: 10.1016/j.diamond.2009.09.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925963509002520.

[3] Q. Wang and I. Balasingham, "Wireless Sensor Networks - An Introduction", in *Wireless Sensor Networks: Application - Centric Design*, 2010, pp. 1–13, ISBN: 978-953-307-321-7. DOI: 10.5772/13225. [Online]. Available: http://cdn.intechweb.org/pdfs/12464.pdf.

[4] C.-Y. Chong and S. P. Kumar, *Sensor networks: evolution, opportunities, and challenges*, 2003. DOI: 10.1109/JPROC.2003.814918.

[5] D. C. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole, "Research Challenges in Environmental Observation and Forecasting Systems", in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00, New York, NY, USA: ACM, 2000, pp. 292–299, ISBN: 1-58113-197-6. DOI: 10.1145/345910.345961. [Online]. Available: http://doi.acm.org/10.1145/345910.345961.

[6] P. Padhy, K. Martinez, A. Riddoch, H. L. R. Ong, and J. K. Hart, "Glacial Environment Monitoring using Sensor Networks", *RealWSN*, pp. 10–14, 2005. [Online]. Available: http://eprints.soton.ac.uk/260845/.

[7] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of Things—A survey of topics and trends", *Information Systems Frontiers*, Mar. 2014, ISSN: 1387-3326. DOI: 10.1007/s10796-014-9489-2. [Online]. Available: http://link.springer.com/10.1007/s10796-014-9489-2.

[8] G. Trickey, "GSMA: Driving Innovation in Connected Living", GSMA, Tech. Rep., 2014. [Online]. Available: http://www.gsma.com/newsroom/wp-content/uploads/us-m2m-2014.pdf.

[9] Machina Research, “The Global M2M Market in 2013 WhitePaper”, no. January, 2013. [Online]. Available: `http://www.telecomengine.com/sites/default/files/temp/CEBIT%5C_M2M%5C_WhitePaper%5C_2012%5C_01%5C_11.pdf`.

[10] Gsma Intelligence, “From concept to delivery: the M2M market today”, no. February, pp. 1–21, 2014. [Online]. Available: `https://gsmaintelligence.com/files/analysis/?file=140217-m2m.pdf`.

[11] OneM2M, *oneM2M Why Join?*, 2014. [Online]. Available: `http://www.onem2m.org/whyjoin.cfm` (visited on 10/20/2014).

[12] ETSI M2M TC, “Machine-to-Machine communications (M2M); Functional architecture”, Tech. Rep., 2013, pp. 1–280.

[13] Cocoon, *Cocoon Overview*. [Online]. Available: `http://cocoon.actility.com/documentation/ongv2/overview` (visited on 10/18/2014).

[14] OM2M, *OM2M Architecture*. [Online]. Available: `http://projects.eclipse.org/projects/technology.om2m` (visited on 10/21/2014).

[15] Carriots, *Carriots Ecosystem*. [Online]. Available: `https://www.carriots.com/documentation/carriots%5C_ecosystem`.

[16] OpenMTC, “OpenMTC White Paper - M2M Solutions for Smart Cities and the Internet of Things”, 2013. [Online]. Available: `http://www.open-mtc.org/%5C_files/2013%5C_03%5C_OpenMTC%5C_Whitepaper.pdf`.

[17] Xively, *What is Xively - Xively*. [Online]. Available: `https://xively.com/whats%5C_xively/` (visited on 10/24/2014).

[18] E. Osmanoglu, *Identity and Access Management: Business Performance Through Connected Intelligence*. Elsevier Science, 2013, pp. 47–54, ISBN: 9780124104334. [Online]. Available: `http://books.google.pt/books?id=nW-tAAAAQBAJ`.

[19] C. Messina, *Does OpenID need to be hard?* [Online]. Available: `http://factoryjoe.com/blog/2009/04/06/does-openid-need-to-be-hard/` (visited on 10/26/2014).

[20] OpenID Connect, *OpenID Connect | OpenID*. [Online]. Available: `http://openid.net/connect/` (visited on 10/26/2014).

[21] N. Sakimura, NRI, J. Bradley, P. Identity, M. Jones, Microsoft, B. de Medeiros, Google, C. Mortimore, and Salesforce, *OpenID Connect Core 1.0 (incorporating errata set 1)*, 2014. [Online]. Available: `http://openid.net/specs/openid-connect-core-1%5C_0.html`.

[22] H. Lockhart, B. Campbell, P. Identity, N. Ragouzis, J. Hughes, R. Philpott, E. Maler, S. Microsystems, P. Madsen, and T. Scavo, “Security Assertion Markup Language (SAML) V2.0 Technical Overview”, 2008. [Online]. Available: `https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf`.

[23] P. Mishra, D. Chopra, J. Moreh, and R. Philpott, “Differences between OASIS Security Assertion Markup Language (SAML) V1.1 and V1.0 21”, no. May, pp. 1–6, 2003. [Online]. Available: `https://www.oasis-open.org/committees/download.php/3412/sstc-saml-diff-1.1-draft-01.pdf`.

[24] S. Cantor, J. Kemp, R. Philpott, E. Maler, S. Microsystems, C. P. Cahill, J. Hughes, A. Origin, H. Lockhart, M. Beach, R. Metz, B. A. Hamilton, R. Randall, T. Wisniewski, I. Reid, P. Austel, M. Hondo, M. Mcintosh, T. Nadalin, N. Ragouzis, P. C. Davis, J. Hodges, F. Hirsch, P. Madsen, S. Anderson, P. Mishra, P. Identity, J. Linn, J. Moreh, and A. Anderson, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", *OASIS standard*, no. March, pp. 1–86, 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[25] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, E. Maler, S. Microsystems, C. P. Cahill, J. Hughes, A. Origin, H. Lockhart, M. Beach, R. Metz, B. A. Hamilton, R. Randall, T. Wisniewski, I. Reid, P. Austel, M. Hondo, M. Mcintosh, T. Nadalin, N. Ragouzis, P. C. Davis, J. Hodges, P. Madsen, S. Anderson, P. Mishra, P. Identity, J. Linn, J. Moreh, A. Anderson, and R. Monzillo, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0", *OASIS standard*, pp. 1–46, 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf.

[26] J. Hughes, S. Cantor, and J. Hodges, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2. 0", *OASIS standard*, 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[27] K. Lawrence, C. Kaler, A. Nadalin, R. Monzillo, and P. Hallam-baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", *Security*, vol. 2003, p. 76, 2006, ISSN: 09685227. DOI: 10.1016/S1361-3723(03)03011-2. [Online]. Available: http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[28] K. Lawrence, C. Kaler, A. Nadalin, M. Goodner, and M. Gudgin, *WS-Trust 1.4*, 2009. DOI: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.4519%5C&amp;rep=rep1%5C&amp;type=pdf.

[29] S. Godik, T. Moses, A. Anderson, S. Microsystems, B. Parducci, C. Adams, D. Flinn, G. Brose, H. Lockhart, K. Beznosov, M. Kudo, P. Humenn, S. Andersen, S. Crocker, and P. S. Systems, "eXtensible Access Control Markup Language (XACML)", 2003. [Online]. Available: https://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf.

[30] S. Track and W. Product, "eXtensible Access Control Markup Language (XACML) Version 3.0", no. January, pp. 1–154, 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf.

[31] WSO2, *WSO2 Identity Server Architecture.* [Online]. Available: https://docs.wso2.com/display/IS500/Architecture (visited on 10/28/2014).

[32] MuleSoft, "Evolution of SOA WhitePaper", [Online]. Available: http://docs.media.bitpipe.com/io%5C_10x/io%5C_107089/item%5C_597001/Evolution%20of%20SOA%20Whitepaper.pdf.

[33] G. Flurry and K. J. Clark, "The Enterprise Service Bus, re-examined", pp. 1–22, 2011. [Online]. Available: http://www.ibm.com/developerworks/websphere/techjournal/1105%5C_flurry/1105%5C_flurry-pdf.pdf.

[34] Burton Group and A. T. Manes, “Enterprise Service Bus: A Definition (White Paper)”, pp. 1–35, 2007. [Online]. Available: http://i.i.cbsi.com/cnwk.1d/html/itp/burton%5C_ESB.pdf.

[35] BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, and Sybase, “Service Component Architecture - Building Systems using a Service Oriented Architecture (Joint Whitepaper)”, no. November, 2005. [Online]. Available: http://www.sybase.com/sb%5C_content/1038547/SCA%5C_White%5C_Paper1%5C_09.pdf.

[36] D. Chappell, “Introducing sca”, no. July, 2007. [Online]. Available: http://192.41.170.42/~mdailey/class/sw-arch/Chappell-IntroducingSCA.pdf.

[37] K. Conner and K. Babo, *Next-Generation ESB*, 2010. [Online]. Available: http://www.redhat.com/promo/summit/2010/presentations/jbossworld/developer-insights-ii/wed/kconner-1130-next-generation/summit-jbw-nextgen-es-Final.pdf.