

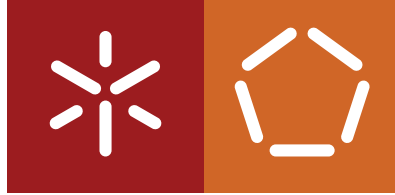


**Universidade do Minho**  
Escola de Engenharia  
Departamento de Informática

Rafaela Cristina Riço Rodrigues

**Mobile Identification as a Service**

March 2022



**Universidade do Minho**

Escola de Engenharia

Departamento de Informática

Rafaela Cristina Riço Rodrigues

## **Mobile Identification as a Service**

Master dissertation

Integrated Master's in Informatics Engineering

Dissertation supervised by

**Professor António Luís Sousa**

March 2022

---

## COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

---

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

LICENSE GRANTED TO USERS OF THIS WORK:



**CC BY**

<https://creativecommons.org/licenses/by/4.0/>

---

## ACKNOWLEDGEMENTS

---

This dissertation is the culmination of five years of hard work, dedication and persistence. However, the success of this journey would not be possible without my support network, to whom I want to show my gratitude.

Firstly, I would like to thank my supervisor, Professor António Luís Sousa, for the availability, guidance and patience through all this years, but especially during the writing of this dissertation.

To Professor Vítor Fonte and Professor João Marco for giving me the opportunity to work with them and for all the support and advice.

To my INESCTEC colleagues that I had the pleasure of working with for all the motivation and help to achieve our goals.

To my friends for making this adventure more fun, rich and for all the support during these five years.

To Maria João and Renata for being the best friends that I could ever ask for.

To the Silva family for treating me as if I were family and for being there in every moments of my life, including the most challenging ones.

To my guardian angels for being the wind beneath my wings.

To my boyfriend Luís, who motivates me every day to be better, for the never-ending encouragement, motivation, love and for being a light in the darkest moments.

To my family, especially my grandparents and godparents, for always rooting for my success.

Finally, the biggest thank you to my parents, for their support, motivation, love and for the sacrifices they made so I could follow my dreams.

My sincere thank you to everyone.

---

## STATEMENT OF INTEGRITY

---

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

---

## ABSTRACT

---

The benefits of using mobile identification applications as substitutes for physical documents are obvious, whether these are university student cards, company employee identification cards, the citizen card or driving license.

However, as these applications grow in popularity and complexity, new requirements and needs arise that need to be addressed without disturbing the normal behavior of the application.

Often the data needed to provide an authentication service is spread across multiple servers, which need to be integrated. This becomes more complicated and complex when an application provides more than one form of authentication (a driving license and a student card require data provided by different services).

In this dissertation we are going to look for solutions that allow to develop an architecture that is prepared to integrate new services at runtime and allows the management of the system, maintaining its dynamic and independence from third parties, regardless of the technology and form of communication used by them.

So, this dissertation presents the state of the art regarding the integration of multiple service providers and the design and implementation a proposed solution, using the WSO2 products to do so. This process is performed in the context of the mobile ID, that is a implementation of a mobile driving license based on the ISO/IEC 18013-5:2021.

**KEYWORDS** mobile identity, digital identity, SIAM, SOA, ESB, Service, Integration, WSO2, Mobile ID.

---

## RESUMO

---

São cada vez mais evidentes os benefícios do uso de aplicações de identificação móvel como substitutos aos documentos físicos, sejam estes cartões de estudantes universitários, cartões de identificação de funcionários de empresas, o cartão de cidadão ou a carta de condução.

No entanto, à medida que estas aplicações se tornam mais populares e mais complexas, surgem novas exigências e necessidades que precisam de ser colmatadas sem perturbar o normal funcionamento da aplicação.

Muitas vezes os atributos necessários para fornecer um serviço de identificação encontram-se distribuídos por múltiplos servidores, que necessitam de ser integrados. Isto torna-se mais complicado e complexo quando uma aplicação disponibiliza mais de uma forma de identificação (uma carta de condução e um cartão de estudante requerem dados fornecidos por múltiplos e diferentes serviços).

Nesta dissertação vamos procurar soluções que permitam desenvolver uma arquitetura que esteja preparada para integrar novos serviços em runtime e permitir toda a gestão do sistema, mantendo a aplicação dinâmica e independente de entidades terceiras, independentemente da tecnologia e forma de comunicação usada pelo serviço.

Assim, nesta dissertação é apresentado o estado da arte relativamente à integração de múltiplos fornecedores de serviço e o design e implementação da solução proposta, utilizando os produtos do WSO2 para fazê-lo. Todo este processo é realizado no contexto do mobile ID, que é uma implementação da carta de condução digital baseada na ISO/IEC 18013-5:2021.

**PALAVRAS-CHAVE** identidade móvel, identidade digital, SIAM, SOA, ESB, serviços, integração, WSO2, Mobile ID

---

# CONTENTS

---

## Contents [iii](#)

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Objectives	1
1.2	Document Structure	2
<b>2</b>	<b>STATE OF THE ART</b>	<b>3</b>
2.1	Service Integration and Management	3
2.1.1	What is Service Integration and Management?	3
2.1.2	The SIAM Ecosystem	4
2.1.3	SIAM Roadmap	7
2.1.4	SIAM Structures	9
2.1.5	Why SIAM?	12
2.2	Service Oriented Architecture	12
2.2.1	Service Oriented Architecture Concepts	13
2.2.2	How to compose a SOA	14
2.2.3	Message Exchange Patterns	15
2.2.4	SOA-Based Architecture Models	17
2.2.5	Benefits of a Service Oriented Architecture	20
2.2.6	Challenges of service-orientation	20
2.3	Enterprise Service Bus	21
2.3.1	Enterprise Service Bus responsibilities	22
2.3.2	Enterprise Service Bus designs	23
2.3.3	Existing solutions	26
<b>3</b>	<b>ARCHITECTURE AND WSO2 INTEGRATION</b>	<b>29</b>
3.1	Architecture	29
3.2	Integration with WSO2	31
3.2.1	WSO2 Enterprise Integrator	31
3.2.2	WSO2 Integration Studio	31
3.2.3	Integration Project Structure	33
3.2.4	ESB Flow	36
3.2.5	Configuration	38



3.2.6	Build and Run	39
<b>4</b>	<b>IMPLEMENTATION</b>	<b>42</b>
4.1	REST API Implementation	42
4.2	Authentication using Chave Móvel Digital	44
4.2.1	Planning	45
4.2.2	Implementation	46
4.3	Attributes from IMT	49
4.3.1	Planning	49
4.3.2	Implementation	50
4.4	LDAP	52
4.4.1	Planning	53
4.4.2	Implementation	53
4.5	Implementing the solution	54
<b>5</b>	<b>INTEGRATION WITH MOBILE ID</b>	<b>56</b>
<b>6</b>	<b>CONCLUSION</b>	<b>59</b>
6.1	Final considerations	59
6.2	Future Work	60
<b>I</b>	<b>APPENDICES</b>	
<b>A</b>	<b>WSO2 ENTERPRISE INTEGRATOR SOURCE CODE</b>	<b>64</b>
a.1	MobileIdentificationApi	64
a.2	FaultSequence Template	65
a.3	Chave Móvel Digital Resource	65
a.3.1	Chave Móvel Digital Sequence	65
a.3.2	AMA Endpoint	67
a.4	IMT Resource	68
a.4.1	IMT Sequence	68
a.4.2	IMT Data Mapping	70
a.5	LDAPUniversity Resource	72

---

## LIST OF FIGURES

---

Figure 1	SIAM layers	4
Figure 2	Customer organization layers and consumers of service	5
Figure 3	Service Providers	6
Figure 4	Example of a service model with service hierarchy	7
Figure 5	Externally sourced service integrator	9
Figure 6	Internally sourced service integrator	10
Figure 7	Hybrid sourced service integrator	11
Figure 8	Request/Response message exchange pattern [16]	15
Figure 9	One-way message exchange pattern [16]	16
Figure 10	Example of two one-way message exchange pattern [16]	16
Figure 11	Logical SOA-based architecture model [16]	18
Figure 12	SOA-based architecture model with logical and technical aspects [16]	19
Figure 13	Technical SOA-based architecture model [16]	20
Figure 14	Architecture with point-to-point integration	21
Figure 15	Architecture with an ESB	22
Figure 16	An ESB providing point-to-point connections	24
Figure 17	An ESB mediating connections	24
Figure 18	An ESB using interceptors	25
Figure 19	Connecting to a protocol-driven ESB	25
Figure 20	Connecting to an API-driven ESB	26
Figure 21	Proposed Architecture	30
Figure 22	Integration Studio getting started page	32
Figure 23	Integration Studio view	33
Figure 24	Menu for selecting the modules	33
Figure 25	Menu for selecting the artifacts to deploy	35
Figure 26	Flow of an ESB-based integration	36
Figure 27	Mediation Sequence	36
Figure 28	Mediation Sequence after an Error	37
Figure 29	Deployment.toml file snippet	38
Figure 30	Pom.xml file in the WSO2 IS	39
Figure 31	Composite Exporter dropdown	39
Figure 32	Open Monitoring Dashboard button	40
Figure 33	Dashboard view	41

Figure 34	REST API creation	42
Figure 35	FaultSequence design	44
Figure 36	Visual implementation of the CMD use case	49
Figure 37	Mapping	52
Figure 38	Visual implementation of the IMT use case	52
Figure 39	mDL components	56
Figure 40	Issuing authority architecture before using the proposed solution	57
Figure 41	Issuing authority architecture after using the proposed solution	58

---

## LIST OF TABLES

---

Table 1	Comparison between tight and loose coupling [16]	14
---------	--	----

---

## LIST OF ABBREVIATIONS

---

- AMA** Agência para a Modernização Administrativa. 44
- API** Application Programming Interface. 12, 17, 25, 26, 30, 31, 32, 34, 38, 43, 45, 46, 47, 50, 54, 55, 57, 60
- BPM** Business process modeling. 15, 19, 20
- CAR** Component Application Resource. 35, 40
- CLI** Command Line Interface. 26
- CMD** Chave Móvel Digital. iv, 42, 44, 45, 46, 47, 48, 54, 55, 57, 59
- CVS** Comma-separated Values. 32, 51
- DN** Domain name. 53
- ESB** Enterprise Service Bus. v, 2, 3, 12, 14, 17, 19, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 34, 35, 36, 57, 59
- HTTP** Hypertext Transfer Protocol. 12, 13, 26, 27, 30, 32, 34, 36, 43, 45, 46, 47, 48, 54
- HTTPS** Hypertext Transfer Protocol Secure. 26, 36
- IMT** Instituto da Mobilidade e dos Transportes Terrestres. 1, 42, 49, 51, 55, 57, 59
- IRN** Instituto dos Registos e Notariado. 1
- ISO** International Organization for Standardization. 50, 57
- IT** Information Technology. 3, 5, 7, 9, 20, 22
- JDBC** Java Database Connectivity. 27
- JMS** Java Message Service. 26, 27, 34, 36
- JMX** Java Management Extensions. 28
- JSON** JavaScript Object Notation. 13, 30, 38, 43, 44, 45, 46, 49, 50, 51, 52, 53, 55, 60
- LDAP** Lightweight Directory Access Protocol. 38, 42, 52, 53, 54, 59
- mDL** Mobile Driver License. vi, 50, 55, 56, 57, 60
- MI** Micro Integrator. 26, 31, 34
- MIAS** Mobile Identification as a Service. 42
- NIC** Civil Identification Number. 49
- NIF** Tax Identification Number. 49
- OTP** One-time Password. 45

- POJO** Plain Old Java Objects. 27
- REST** Representational State Transfer. 30, 32, 38
- SIAM** Service Integration and Management. v, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 59
- SOA** Service-Oriented Architecture. 2, 3, 12, 13, 14, 15, 17, 20, 21, 59
- SOAP** Simple Object Access Protocol. 12, 13, 25, 32, 38, 49, 51
- TCP** Transmission Control Protocol. 34
- TOML** Tom's Obvious, Minimal Language. 38
- UK** United Kingdom. 3
- URL** Uniform Resource Locator. 34, 48, 51, 54, 55
- VFS** Virtual File System. 36
- WSDL** Web Service Description Language. 27, 34, 35, 49, 51
- WSO2 EI** WSO2 Enterprise Integrator. 26, 27, 28, 31, 33, 34, 38, 40, 59
- WSO2 IS** WSO2 Integration Studio. v, 31, 32, 39, 59
- XML** Extensible Markup Language. 12, 27, 38, 51, 60
- XPath** XML Path Language. 43, 44, 46
- XSTL** Extensible Stylesheet Language. 35

---

## INTRODUCTION

---

With the advances in technological development, the need for digital transformation in certain areas is becoming evident. One of these areas is the mobile digital identity. Recent analysis from the Juniper Research estimates that the growth in solutions of this area could exceed 800% over the next 5 years and mobile phones will become the primary source of identity for over 3 billion people by 2024. This is the reason why companies are turning to mobile by default [24].

Digital identity sits at the heart of economic and social transformation. Around the globe, governments are trying to implement solutions that allow them to identify the citizens in addition to the traditional ones (physical cards) [14]. For example, the `id.gov.pt` is an app created by the Portuguese government that allows the user to save and consult his documents like the citizen card, drivers licence and health cards in the mobile phone.

As the market evolves over the coming years, it will move rapidly from a 'card first and mobile companion' landscape to a world where digital takes precedence. The physical card or document will still be with us, but it will move to support the digital tokens used to manage proof of identity [14]. This can include university student cards, company employees cards and every other form of identification.

However, with the growth of this type of applications in popularity and complexity, new requirements and needs arise that need to be addressed without disturbing the normal functioning of the application. One issue is related to the information necessary to provide a mobile identification service. Usually the data is stored in multiple databases from different providers and it is necessary to integrate the external service in the applications without stopping the service, adding a new level of complexity. For example, in Portugal, to access all the data of a driving license, it is necessary to interact with different sources of attributes, such as the [Instituto dos Registos e Notariado \(IRN\)](#), [Finanças](#) and [Instituto da Mobilidade e dos Transportes Terrestres \(IMT\)](#) and then merge them.

This is an example of the services that must be interacted with in the digitization of the drivers license. Depending on the application being developed, other services may be used. Developing and implementing an architecture prepared to deal with such scenarios is the main motivation on this dissertation

### 1.1 OBJECTIVES

The purpose of this master thesis is to develop an architecture that integrates new services at runtime and allows the entire management of the system, maintaining a dynamic and independent application from third parties and

the information provided by them. The application is going to work like a wallet, so it needs to be able to expand and adapt to new providers.

In order to achieve the main goal, is necessary to:

- Design and develop a service-oriented architecture for mobile identity apps
- Integrate sources of external attributes: adding, removing and changing any service, whether external or internal to the system.
- Create and manage attributes and namespaces.

If all of these objectives are fulfilled, it guarantees the management of the system's life cycle, which is the main objective of this master thesis.

## 1.2 DOCUMENT STRUCTURE

This document consists in 6 Chapters and is divided in the following structure:

- **Chapter 2:** presentation of the current state of the theme addressed in this dissertation, namely the concept of [Service Integration and Management \(SIAM\)](#), [Service-Oriented Architecture \(SOA\)](#) and [Enterprise Service Bus \(ESB\)](#).
- **Chapter 3:** presentation of the proposed architecture and the set of components provided by WSO2 for implementing the solution.
- **Chapter 4:** implementation of the proposed architecture with 3 use cases included.
- **Chapter 5:** integration of the solution with mobile ID using the mobile driver license as a proof of concept.
- **Chapter 6:** conclusions about the dissertation work and future improvements.



---

## STATE OF THE ART

---

This Chapter presents the outcome of the research about the current state of the market and possible solutions for the problem under study. We will start by presenting the concept of [Service Integration and Management \(SIAM\)](#) and explaining why is it useful in this situation. Then we will explore the concepts and components of a [Service-Oriented Architecture \(SOA\)](#). At last, we will focus on the [Enterprise Service Bus \(ESB\)](#), one of the most important components of a [SOA](#), including solutions available in the market.

### 2.1 SERVICE INTEGRATION AND MANAGEMENT

In the past, [IT](#) organizations did most of their work internally, from racking servers to app development. However, with the current pace of technology innovation, organizations are being forced to resort to external expertise more regularly [13].

This lead to an increased use of outsourcing services in multiple areas such as data, security, and even functionalities.

But how do we maintain an end-to-end overview with numerous suppliers involved? And how do we keep the quality of service when working with both external and internal services? [13]

So, [SIAM](#) was created in response to these type of questions. Using industry terms, [SIAM](#) aims to provide [IT](#) organizations with the ability to establish a mechanism to manage the complexity of integration service providers in a multi-sourced environment [26]. They want to facilitate working with many different suppliers in a smarter and more organized approach [13].

#### 2.1.1 *What is Service Integration and Management?*

The term [SIAM](#) and the concept of Management methodology was originated around 2005 inside the [UK](#) government. The methodology was first designed for the Department of Work to improve the value for money in services delivered by multiple providers and separate service integration capabilities from systems integration and [IT](#) service provisions [25].

With this approach, which is seen as a methodology, in addition to the number of activities in the providers being reduced to half, it introduced the concept of a "service integrator", which will be addressed soon [25].

In 2015, this concept gained more visibility and interest after AXELOS<sup>1</sup> published several white papers on SIAM. A year later, Scopism<sup>2</sup> formed the SIAM Foundation Architect Group to gather practitioners of this concept and create a consolidated view of their knowledge and experience. The success of the SIAM Foundation Body of Knowledge in 2017 was followed by the subsequent release of the SIAM Professional Body of Knowledge in 2018, providing further insight and guidance in the application of SIAM practices [26].

So, the SIAM Foundation Body of Knowledge [25] defines SIAM as:

"A management methodology that can be applied in an environment that includes services sourced from a number of service providers."

This concept has a different level of focus to traditional multi-sourced ecosystem. It supports cross-functional, cross-process and cross-provider integration. Another particularity about the SIAM is that it creates an environment where all parties know their role, responsibility and context in the ecosystem, are empowered to deliver and are responsible for the outcome.

Also, SIAM presents the concept of Service Integrator, which groups and consolidates the providers in order to delivery a reliable and cohesive collection of services, providing effective governance and management of a multi-service-provider ecosystem [17].

This methodology can be adopted by different sizes and types of organizations and industry sectors. However, for costumers with single providers, SIAM is unlikely the best solution [25].

### 2.1.2 The SIAM Ecosystem

The SIAM ecosystem consists in tree layers: Customer Organization, Service Integrator and Service Provider. Figure 1 presents a simple diagram of the SIAM model.

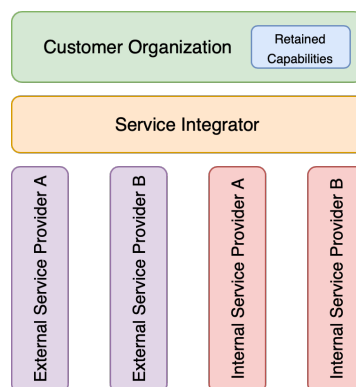


Figure 1: SIAM layers

1 <https://www.axelos.com>

2 <https://www.scopism.com>

### Customer Organization

The customer organization is the end client that usually includes business units such as human resources, sales and internal IT functions. In addition, they have their own costumers who use their services and owns the contractual relationships with external services and integrator [25].

The organization will include some retained capabilities, which are functions that are responsible for architectural, strategic, business engagement and corporate governance activities. Some examples are enterprise architecture, governance of the service integrator and contract management [25].

The service integrator is independent from the retained capability.

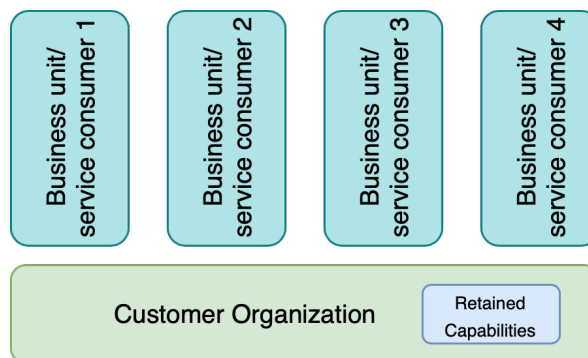


Figure 2: Customer organization layers and consumers of service

### Service Integrator

The service integrator layer is a single, logical entity responsible for the end to end service, governance, management, integration and coordination between the Customer organization and the service providers. It focus is implementing an efficient cross-service provider organization, making sure that all service providers are contributing to the end to end service [25].

This layer removes the obligation for the customer to concern itself with the management overhead of looking after the complex web of service providers [26].

### Service Providers

The service providers are responsible to deliver one or more services to the customer. This implies being responsible for the technology used in the contract and operating their own processes [25].

The service providers can be:

- Internal - delivered by a team or department that is part of the customer organization. Usually is regulated by agreements and targets. If the customer retains its own IT capability, it is considered an internal service.
- External - services are provided by an organization that does not belong to the service organization. They are regulated by contracts.

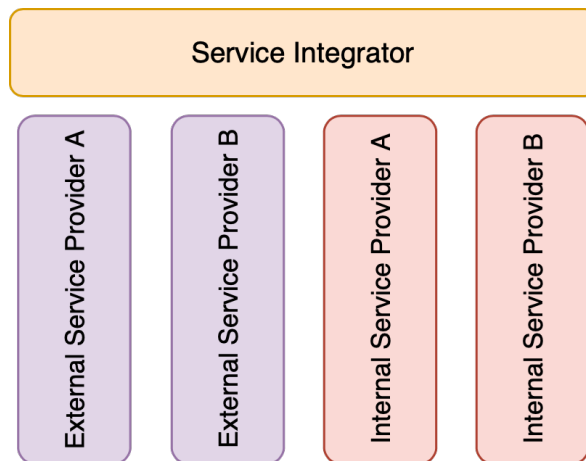


Figure 3: Service Providers

Examples of services provided by service providers includes: Data center, security, network, cloud services.

#### *SIAM Scope*

Although the scope of **SIAM** will vary from organization to organization, all of them need to define the service(s) that are in scope.

So, for each service, these are the areas that need to be defined:

- Service outcomes, value and objectives
- The service providers
- The service consumers
- The service characteristics (which includes service levels)
- The service boundaries
- Dependencies with other services

Also, it is imperative to create a service model that displays the hierarchy of services, as we can observe in the Figure 4.

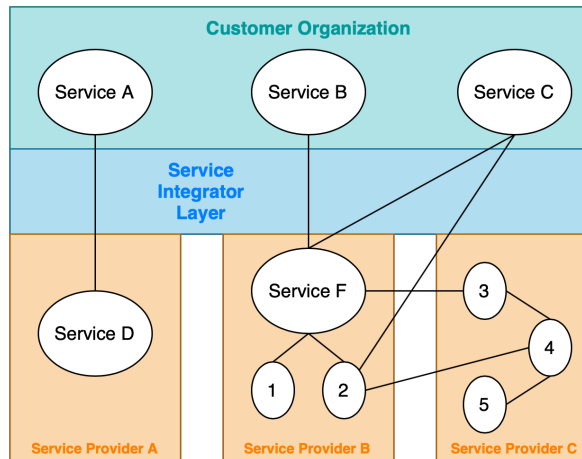


Figure 4: Example of a service model with service hierarchy

### 2.1.3 SIAM Roadmap

The SIAM roadmap is a plan for the implementation of the SIAM as part of operating model of an organization [25].

Using a roadmap for the implementation helps the organization in defining the SIAM requirements, provides a planning framework, helps determining the most appropriated structures and model and leads the implementation [25].

The roadmap consists of four high-level stages, which can be executed iteratively:

- Discovery and Strategy
- Plan and Build
- Implement
- Run and Improve

Each stage has their objectives, triggers, inputs, activities and outputs. In the end of the each stage, there is a checkpoint to review the outputs, risks, issues and the plan for the next stage [25].

#### *Discovery and Strategy*

Discover and Strategy initiates the SIAM transformation project, maps the current situation and formulates key strategies. This stage can be triggered, for example, by the desire or necessity of the company to move to a SIAM ecosystem [19].

This stage receives as input all informations and data about the state of the business like the current procurement, IT strategies, organization structure, products and practices [25].

The objectives for this stage are:

1. Establish the **SIAM** transition project
2. Establish a governance framework
3. Define the strategy and outline model for **SIAM** and the services in scope
4. Analyze the current state of the organization, including skills, service, service providers, tools and processes
5. Analyze the marketplace for potential service providers and service integrators. [25]

#### *Plan and Build*

The Plan and Build stage is initiated after the Discovery and Strategy stage is finished and the organizations confirms their intention to continue with a **SIAM** implementation [25].

This stage, that is build on top of the output of the previous stage, is initiated when the detailed design for **SIAM** is complete and the plans for the move to a new operating approach is created [25].

The main objectives for this step are:

1. Complete the design of the **SIAM** model, including the services that are in scope
2. Obtain full approval for the **SIAM** model
3. Appoint the service integrator and service providers
4. Commence organizational change management

#### *Implement*

The goal of this stage is to manage the transition to the desired **SIAM** model that will be implemented and in use in the end of this stage, using the output of the Discovery and Strategy and Plan and build stages.

This step is triggered after the previous stages are completed, but the start of the implementation can be influenced by events in the existing environment that the customer organization has limited control of. One example of this situation is the end of a contract of a service provider [25].

#### *Run and Improve*

The last stage, Run and Improve, is triggered when the Implementation phase is completed and the main purposes are to manage the **SIAM** model, day to day service delivery, processes, teams and tools and continuous improvement [19].

It also receives as inputs all the outputs designed by the first three states.

### 2.1.4 SIAM Structures

There are three types of structures in the SIAM ecosystem, which are:

- External Sourced
- Internal Sourced
- Hybrid

The main difference between each structure is the configuration of the service integration layer and the sourcing. The customer organization will choose the structure based on factors, such as: business requirements, internal capacities, budget, current organizational maturity and capability in service integration and IT, among others [25].

#### *External Sourced*

In this structure, the service integrator is provided by a third-party (external organization). Moreover, the external service integrator is exclusively focused on service integration, not behaving as a service provider.

The service providers roles can be performed by an external and/or internal provider.

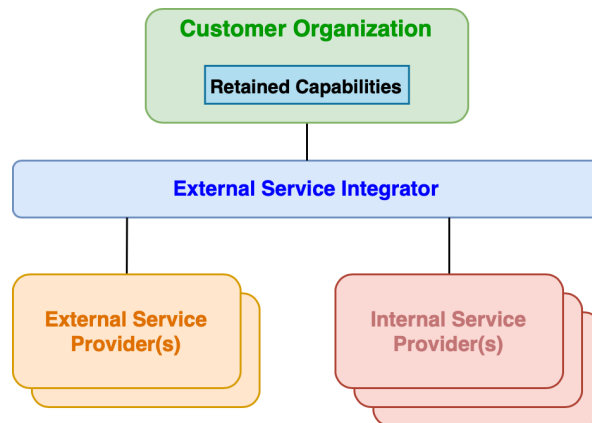


Figure 5: Externally sourced service integrator

This type of structure is suitable for organizations that do not have internal service integration capabilities, do not wish to develop one or the resources available are not enough [25].

This solution implies that the organizations need to have a high degree of trust in the chosen organizations and be prepared for them to assume the role of integrator.

This is an excellent opportunity for the customer to review multiple external services integrators, choosing the best for the organization. Also, this type of organization already has the experience and toolset, allowing the customer to focus on the business outcomes and strategic objectives [25].

On the other hand, sourcing and managing an external organization adds some levels of risk. First, if the customer has not clearly defined their own objectives, the potential for higher costs grows and may lead to a poor service delivery by the external service integrator [25].

Also, the chosen external service integrator’s models and practices may not be the best fit for customer organization and due to the contract established between the organizations, changing how the service integrator works can be challenging [25].

*Internal Sourced*

In the internally sourced service integrator structure it is the customer organization that provides the service integration capability, taking the role of service integrator. Moreover, it is necessary to define and manage the service integrator role and the customer role separately [25].

This structure is highly recommended for organizations that are capable of providing service integration or intends to develop one in-house. This might be because the customer wants to preserve control and flexibility over the SIAM ecosystem or because they are not allowed to use third-parties (the business needs to retain ownership of the service integrator layer or/and timescales do not allow an external service).

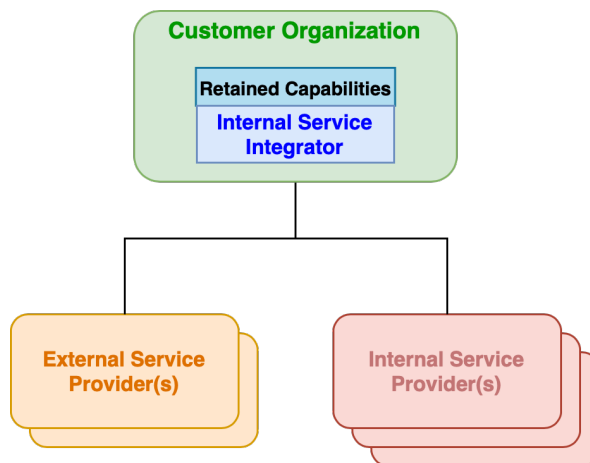


Figure 6: Internally sourced service integrator

This structure gives the customer full control over the service integrator role, without any of the risks and costs from depending on an external company. Owning the service integrator avoids loss of key resources or key knowledge and evades the external services providers to see the integrator as competition (they are more likely to collaborate with him) [25].

Implementing the service integrator can be quicker than expected since it already understands the goals and drivers of the customer organization and since it belongs to the organization that manages the other services providers contracts, it has an advantage over their behaviour and performance. Also, the service integrator might be more flexible and accommodate change without a requirement for any contractual amendments [25].



If the customer organization has no experience implementing a SIAM ecosystem or are not fully committed to it, he may underestimate the amount of resources, skill and expertise needed to build a service integrator that has to be designed, developed and maintained throughout its lifetime [25].

On top of that, having a "in-house" service integrator has included some risks: if there is a conflict between the costumer and the service provider and the service integrator is seen as a synonymous with the customer organization, it would be more challenging for them to mediate. In addition, the internal service providers may not accept the authority of the new internal service integrator [25].

*Hybrid Sourced*

The hybrid service integration is a structure where the customer works with an external organization, which acts as a service integration partner, to create and provide a service integrator. Figure 7 shows the structure of and hybrid integrator.[25]

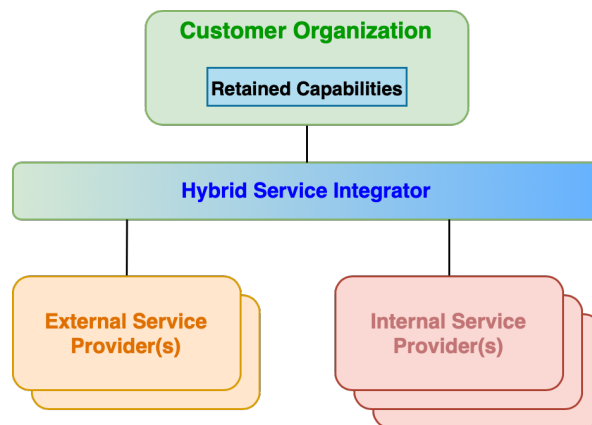


Figure 7: Hybrid sourced service integrator

This structure can be used in a organization temporarily or permanently. If is temporary, the customer can learn from the expertise of the third-party organization and end the hybrid approach when has acquired the skills and expertise and is able to move to an internal service provider, if desired. However, the customer has to pay attention to not build inadvertently a long-time relationship when meant to be temporary [25].

The main difference between the hybrid integrator and the internal integrator is that in the first it is normal to give specific service integration roles, structures and functions to the external organization or the customer [25].

This type of approach allows the benefits to be seen more quickly, makes the transition to a SIAM ecosystem easier and help the customer organization to avoid common mistakes when negotiating, thanks to the experience of the external service integrator [25].

On the down side, the customer must have a service integration capability that if not clearly designed, might led to duplication of skills, roles, toolset and poor service. Moreover, the customer may only want this approach because it is hesitant to give up control, not for business purpose. This can lead to a loss of SIAM benefits [25].

### 2.1.5 Why SIAM?

A successful implementation of a [SIAM](#) ecosystem can provide multiples advantages: better service to the end-users, increased accountability of every service, more flexibility and efficient service management orchestration and delivery [7].

Although the whole concept is very interesting, the purpose of this master thesis is not to fully apply the [SIAM](#) methodology, but some concepts and methodologies that can be used or adapted in order to obtain some of its benefits.

Since we are designing an architecture from scratch, we can apply some of the concepts of [SIAM](#) into our architecture. For example, the three last stages of the [SIAM](#) roadmap are very useful to our purpose. We can start by plan and build our architecture, using some of the activities described in the roadmap Chapter, then implement, run and improve.

Also, the service integrator layer, as observed in [Figure 1](#), works as a intermediate between the customer and the service providers. This methodology is similar to the middleware used in the [SOA](#), the [Enterprise Service Bus \(ESB\)](#).

The [ESB](#) is a pattern where a centralized software component performs integrations to backend systems and makes those integrations available as service interfaces for reuse by new applications. This integration includes translations of data models, deep connectivity, routing, and requests between services. [11]

A proof of this relation is that many [SIAM](#) toolset providers, like [TietoEVRY](#), have created potential integration solutions including the [ESB](#), [Application Programming Interface \(API\)](#) and web service interfaces ([SOAP/HTTP/XML](#)) connectivity through the Internet [23].

So, in the next Sections we will go deeper into the concepts of [SOA](#) and [ESB](#).

## 2.2 SERVICE ORIENTED ARCHITECTURE

Before [SOA](#) has emerged in the late 1990s, the development and integration of applications that needed external data or a functionality available in another system required a complex point-to-point integration, that usually was created or adapted for each project [12].

To overcome that restriction, instead of using monolithic entities, [SOA](#) adopted smaller and manageable blocks - or services. Each service is not an application but components that, when combined in multiple ways, can make an application [27].

However, the real momentum for [SOA](#) was created by Web Services. Although Web Services do not translate to [SOA](#) and [SOA](#) is not based in Web Services, they are mutually influential: Web Services momentum will bring [SOA](#) to mainstream users, and the best-practice architecture of [SOA](#) will help make Web Services initiatives successful. [16]

So, we can define [SOA](#) as a set of design principles that provide structure for the creation of services and how to aggregate them into applications. [SOA](#) also values reusability and agility using service interfaces [12].

### 2.2.1 Service Oriented Architecture Concepts

#### *Drivers*

As business applications grows, they become more and more complex and more involved with other companies applications. However, SOA is well suited to deal with complex distributed systems, as it facilitates interactions between the service providers and consumers, i.e., between the applications [16].

In addition, SOA does not assume that the network of distributed systems are controlled by single owners and it is prepare to deal with multiple owners. Nonetheless, the key to successfully manage numerous owners is compromise between them, accepting that different priorities and solutions exists.

At last, SOA recognizes and accepts that each service might have different attributes, use different platforms and even have different programming languages, not being an impediment to integrate a service provider.

#### *Services*

The Oxford dictionary defines a service as the job that an organization does or a system/organization that provides the public with something that it needs.

In the world of SOA and service-orientation, the term “service” is not generic. It provides a business functionality that is largely self-contained, so it is independent and autonomous. However, some dependence always exists, so the goal is to minimize them [16].

If we focus only in the business aspect, a service works as a 'black box', so it can be viewed in terms of its inputs and outputs, without any knowledge of its internal workings. This means they can be called with little or no knowledge of how the integration is implemented underneath [16].

Technically, each service has the code and data integration necessary for the complete functionality. A service can include (and even be compound entirely) by other services [27].

In addition, since each service only does a certain functionality, it can be used for more than one application and called upon as needed (reusable). However, this can affect the performance, since multiple applications may use the same service simultaneously [16].

#### *Loose coupling*

Loose coupling is the concept used to deal with the requirement of scalability, flexibility and fault tolerance and its objective is to minimize dependencies within and between services. Since it is a principle, every project has it own kind and amount of loose coupling.

In Table 1 are listed some of the topics that can be considered when talking about loose coupling.

#### *Communication*

Since each service is responsible for a functionality, they need to share the results with whoever needs it. So, they usually expose themselves using standard network protocols, such as SOAP or JSON/HTTP, to send requests to read or change data [16].

	<b>Tight Coupling</b>	<b>Loose Coupling</b>
<b>Physical connection</b>	Point-to-point	Via mediator
<b>Communication style</b>	Synchronous	Asynchronous
<b>Data model</b>	Common complex types	Simple complex types only
<b>Interaction Pattern</b>	Navigate through complex object trees	Data-centric, self-contained message
<b>Control of process logic</b>	Central control	Distributed control
<b>Platform</b>	Strong platform dependencies	Platform independent
<b>Deployment</b>	Simultaneous	At different times

Table 1: Comparison between tight and loose coupling [16]

### 2.2.2 How to compose a SOA

After explaining the key technical concepts of **SOA**, it is important to use these concepts in the right way, that is, we need to find the right degree of centralization. To establish it successfully, it is crucial to care about the infrastructure, architecture and processes [16], which will be deepened in the next Sections.

#### *Infrastructure*

The infrastructure of a **SOA** landscape is the technical part that allows high interoperability and is called **Enterprise Service Bus** [16]. The **ESB** promises to build up a **SOA** by iteratively integrating isolated applications into a decentralized infrastructure.

The main feature of an **ESB** is it enables us to call services between heterogeneous entities. In the Section 2.3 this concept will be further developed.

#### *Architecture*

The architecture is what unifies all the **SOA** concepts, **SOA** tools and **SOA** standards in order to design and implement a working and maintainable system [16].

To achieve that, it is necessary to:

- Classify the different types of service
- Decide the level of loose coupling
- Restrict the data model of the service interface
- Define policies, rules and patterns and the infrastructure technology
- Clarify roles and responsibilities

By making these decisions according to our problem, we will build our architecture.

### Processes

Usually there is not a single person or a team responsible for managing everything, which makes the process of going from a business idea to a solution in production mode more complicated and slow.

To avoid this type of situation, it is necessary to define appropriated processes. Processes are a step by step description of the activity or tasks that have to be done to fulfill a certain need.

This processes includes:

- **Business process modeling (BPM)** - breaking up business processes into smaller tasks and activities (in this case are services)
- **Service lifecycle** - different steps a service needs to take to become reality
- **Model-driven software development** - generate code for dealing with services

### 2.2.3 Message Exchange Patterns

Since there are different ways to exchange messages between two services or application, the Message Exchange Pattern defines the sequence of messages in a service call or service operation, specifying the order, direction, and cardinality of those messages [16]. Next Sections will present some of the most important patterns in SOA.

#### Request/Response

This pattern is one of the most important patterns used in SOA. The consumer sends a message to the service provider and waits for an answer, that can contain data or simply a confirmation of the successful processing of the request [16].

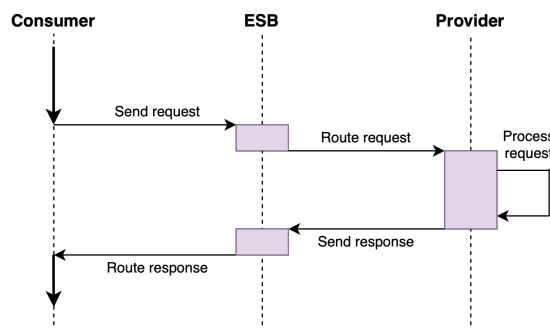


Figure 8: Request/Response message exchange pattern [16]

The drawback of this pattern is that it is blocking, meaning that while the consumer waits for a response, he cannot do anything else. This means that the consumer needs a fast response or running time is not crucial.

However, if the provider has a problem and is not available, the consumer can be forever in a blocked state. To solve this, a timer that triggers an exception if the consumer does not receive a response within a certain amount of time can be added [16].

One of the big advantages of using this pattern is the simplicity of the code. If the consumer needs information from another service, it can continue the work after receiving a response/acknowledgment, knowing that the necessity has been dealt with [16].

*One-way*

This pattern is used when a service does not need a response for the request sent.

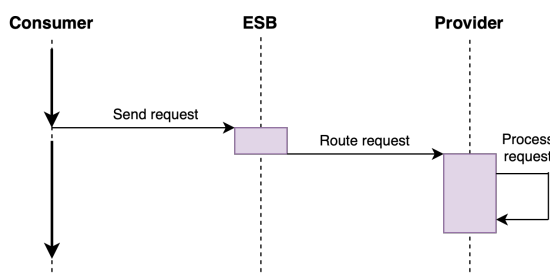


Figure 9: One-way message exchange pattern [16]

We can argue that the request/response pattern is a aggregation of two one-way pattern. However, a combination of two one-way could lead to an asynchronous and non-blocking communication, since the consumer would not wait for the response and the sender of the first request (consumer) would also have to be a provider to receive the request from the other service [16].

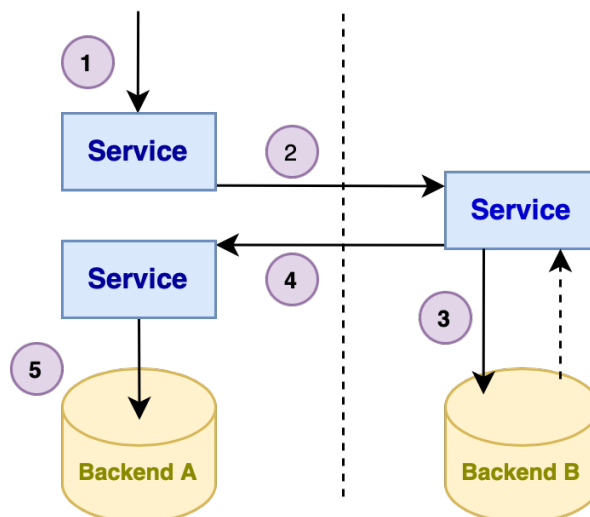


Figure 10: Example of two one-way message exchange pattern [16]

### *Request/Callback*

If the process/thread needs some data or confirmation but does not need to wait for it, the Request/Callback is the one to be used. This pattern is also called asynchronous request/response [16].

Conceptually, a consumer's API in this case, works such that the consumer initiates a request and specifies what to do when the answer arrives. Technically, the consumer might define a callback function, which is the method that is called when the response arrives [16].

The biggest benefit of this pattern is that, thanks to the asynchronicity, it introduces a form of loose coupling, since the provider does not need to be available when the message is sent and the consumer can continue to work.

On the other hand, working with asynchronous responses can lead to some complications:

- If more than one message is sent, the responses may be received in different order from the one that were sent, so the service has to be able to deal with this. Usually each message has a correlation ID that is passed in the message and the response.
- The context for each response has to be valid and has all the information required to process the response
- Some responses may never arrive
- The response may not be delivered to the same process/thread

### *Publish/Subscribe*

Sometimes one-way messages are only used to inform another system that something has changed. These kind of messages are called notifications or events.

This pattern, in general, allows multiple systems (or observers) to "subscribe" another system so that it will notify them when a specific condition occurs (publishers) [16].

In this pattern, instead of delivering the message directly to the observer, the published messages are categorized into classes without knowledge of which subscribers, if any, there may be. This is achieved using brokers.

Brokers are known by both publisher and subscriber. The publisher will send the message to the message broker and the message broker will filter and broadcast the message to the right subscriber. In SOA, the broker is the ESB [16].

#### 2.2.4 SOA-Based Architecture Models

The diagrams of a SOA-based architecture can take many forms, depending on the point of view and intentions of the illustrator. The illustrator may focus on business, domain, logical or technical aspects.

### Logical Architecture Models

This model, as shown in Figure 11, offers a business or logical view of the aspects of a system.

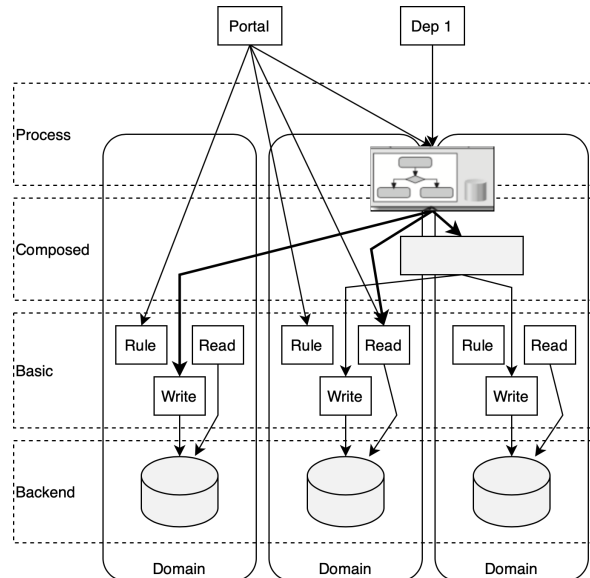


Figure 11: Logical SOA-based architecture model [16]

Inside every organization we can find departments and each department plays a specific role and has distinct functionalities. In the landscape, these are called domains. [16]

Each domain is responsible for a backend and for provide basic services. The backend in this model represent is technical system and the data and capabilities associated with it. The basic services wrap the technical details of the backend and are divided in three types: reading and write services and rule services (logic services that provides general business rules) [16].

On the other hand, the composed and process services do not belong to a single domain, since they combine multiple business functionalities. However, someone needs to be responsible for them [16].

The composed services presents a short-running flow of services inside the business process. They operate in a higher level than basic services, but they also are short-term running and stateless [16].

One solution is creating a cross-domain department. This type of departments, being a domain itself, is responsible for the overall processes from multiples "natural" domains.

In the Figure 11 is also visible the existence of hierarchies of domains. For example, suppose that one domain is formed by two composed services and a subdomain with a basic service. The first composed service would use the basic service inside the domain and the second composed service would also use services from another domain [16].



### Mixed Architecture Models

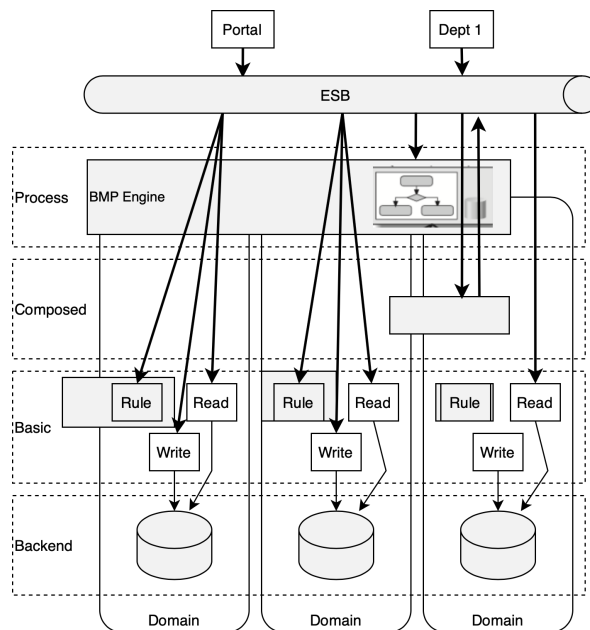


Figure 12: SOA-based architecture model with logical and technical aspects [16]

Figure 12 presents the same diagram as the Figure 11 but with more technical details. The three new components inserted in the architecture are:

1. The **ESB**, through which all messages are routed.
2. The processes which can be implemented by a **BPM** engine or tool.
3. All basic logic services that provides business rules which are implemented inside a rules engine

Despite the new components, the architecture remains domain-driven and does not contradicts the first one (as they are the result of different points of view).

### Technical Architecture Models

At last, if we are only interested in the technical view, the architecture might look like the one in the Figure 13.

Since we are focused on the technical aspects of the system, the **ESB** has more prominence. The domains are only responsible for basic and composed services and the logic services and process services are separated, since special tools are provided for them [16].

While logically a rule service belongs to an ordinary business domain, it might be better to manage all the rules in a common domain associated with a rules engine. Likewise, defining all the process services in a common central place may be desirable [16].

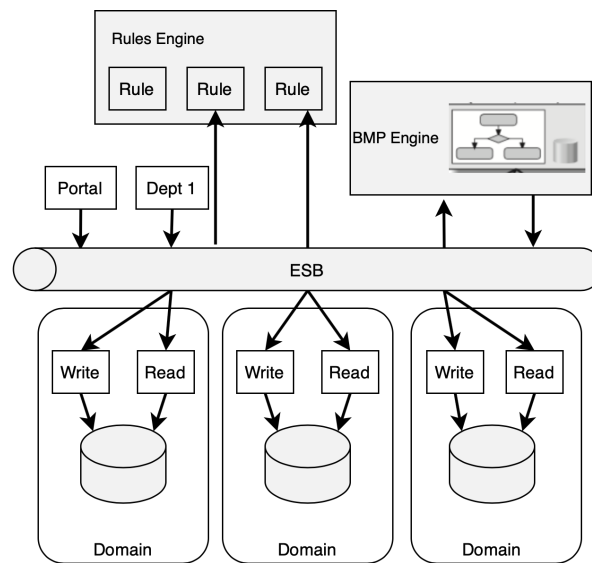


Figure 13: Technical SOA-based architecture model [16]

### 2.2.5 Benefits of a Service Oriented Architecture

The **BPM** offers multiple benefits for the organizations [12] [27]:

- Greater business agility;
- Reduction of the cost and complexity of the enterprise software development;
- Reduced time to market. Since the applications are developed with reusable service interfaces and functionality, developing new applications that uses those are faster to be implemented and deployed;
- The **IT** department is more responsible to its needs, leading to a more agile organization;
- Improved collaboration between business and **IT**;
- Ability to leverage legacy functionality in new markets: multiple organizations have used **SOA** to exposed functionalities to the web, allowing their clients to serve themselves to the information without a direct interaction with the company's employees

There are several companies that adopt **SOA**, including: Pfizer, Bank of America, jetBlue, BNSF Railway and Merrill Lynch [10].

### 2.2.6 Challenges of service-orientation

Although **SOA** has many benefits that were described in the previous Section, it also has some challenges that needs to be addressed.

First, when a company is implementing it, it requires an increased organizational discipline by every member involved in the process [18]. This can be challenging if an organization does not have every role well defined and every process clear.

Another issue is related with interoperability. The interdependence between services in a SOA architecture is highly discouraged. However, we need to balance the level of dependency and the compatibility between services because a low interdependence can lead to incompatibility between services. [18].

A service based architecture involves services interacting with each other through messages. This implies that an architecture can have tons of services and that each service can produce thousands or millions of messages. In a situation like this, managing these many services can become very difficult, especially when there are external service providers [18].

Business related, SOA does not guarantee a reduction of costs and does not promise improvements in the agility of the system. Therefore, the expectations needs to be realistic [18].

At last, one of the challenges in SOA is the lack of testing space. The solution is to invest in a testing framework that could test headless services like messages or database services [18]. This creates another expense for the company.

### 2.3 ENTERPRISE SERVICE BUS

As mentioned in the previous Section, the Enterprise Service Bus (ESB) is an essential component of a SOA infrastructure.

Before the ESB, business had to do point-to-point application integration and create custom-code for every application. This lead to insecure, non-monitorable and often non-manageable communication channels between tightly-coupled applications. Today, the ESB eliminates these challenges by serving as a stable communication bus between applications. [8]

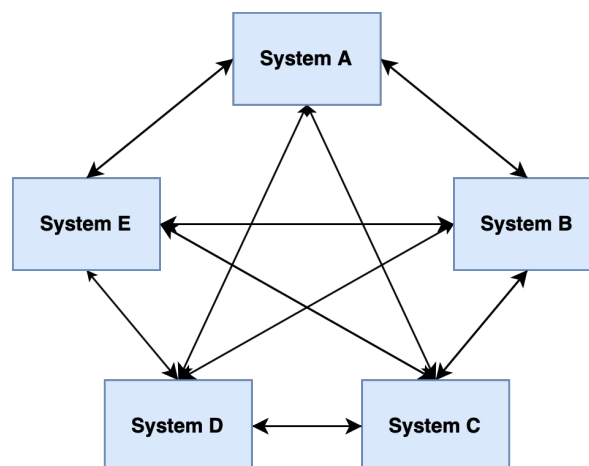


Figure 14: Architecture with point-to-point integration

An **Enterprise Service Bus** is defined as an architectural pattern through which a centralized software component, the bus, integrates multiple applications [11]. Using transformations, message routing and protocol conversions, the **ESB** decouples systems from each other, allowing them to communicate without dependency or knowledge of the other systems. These integrations and transformations are available as a service interface for reuse by new applications [20]. By using all of these components in a **ESB**, it provides a safe, reliable and fast backbone to your **IT** infrastructure [15].

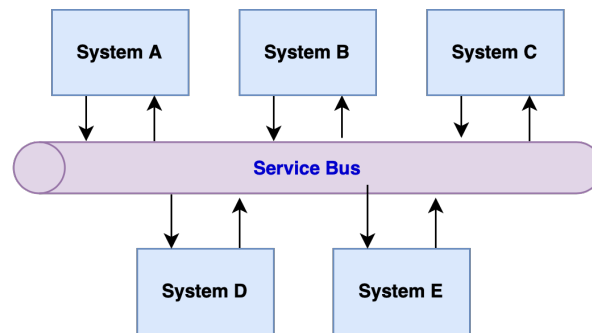


Figure 15: Architecture with an **ESB**

### 2.3.1 Enterprise Service Bus responsibilities

As was previously said, the main role of an **ESB** is provide interoperability. So, in this Section we will cover the collection of characteristics and responsibilities that allow the **ESB** to fulfill its promises.

#### *Data transformation*

A crucial element of the **ESB** is the data transformation. Since we are integrating multiples systems that may communicate in various formats, the bus transforms the incoming and/or the outgoing message and data for communicating with the applications.

As **Chappell** states:

"Data transformation is inherently part of the bus in an **ESB** deployment. Transformation services that are specialized for the needs of individual applications plugged into the bus can be located anywhere and accessible from anywhere on the bus. Because the data transformation is such an integral part of the **ESB** itself, an **ESB** can be thought of as solving the impedance mismatch between applications."

#### *Intelligent routing*

Since the **ESB** prevent applications to communicate directly with each other, it needs to implement mechanisms to delivery messages from a consumer to the provider and sometimes deliver a response.

Depending on the technology and the level of intelligence provided, this task may require different levels of processing. [16]

Some mechanisms are message brokers that can be based on the payload schema or the content of the request; publish-subscribe messaging that makes the process of adding, remove and subscribe the users easier and the failover routing, that can redirect messages if the service is unavailable or failing.

### *Reliability*

Different protocols offer different forms of reliability. Some protocols do not guarantee that the message was delivered, while others guarantee that the message will be delivered at least one time. This is the reason why the **ESB** must define how it will deal with faults, timeouts and technical errors, to ensure the stronger level of reliability possible. [16]

### *Monitoring and logging*

**ESB** becomes the center of your running business processes, and the way you implement, debug, and maintain local processes is now distributed over multiple systems [16]. Therefore, it is important to integrate tools that supports monitoring and debugging. This will allow a rapid response if a problem occurs and allows you to follow the entire process closer.

## 2.3.2 Enterprise Service Bus designs

Technically and conceptually, **ESBs** can vary substantially. It might simply define a protocol, delegating the majority of the tasks to the consumers and providers or it might consist of several tools and programs working together. [16]

So, in this Section we will explore different technical approaches for implement an **ESB**.

### *Point-to-Point Connections Versus Mediation*

One of the aspects that differs in an **ESB** implementation is the amount of coupling that provides for the connections.

The point-to-point connection (Figure 16) is used when the consumer needs to know the endpoint of the service providers. The problem with this type of connection is that the call fails if the physical receiver is not available [16].

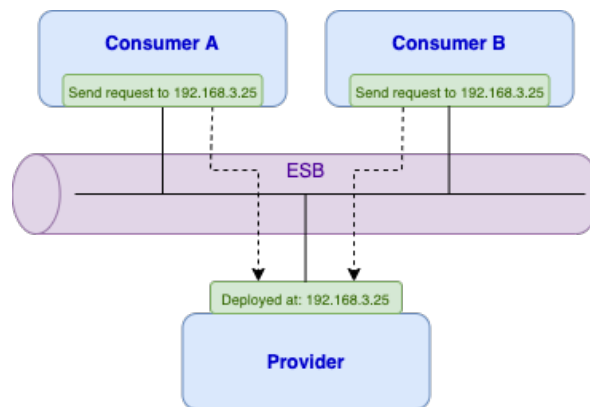


Figure 16: An ESB providing point-to-point connections

When the consumer does not need to know the endpoint of the provider, the ESB plays the role of a mediator or broker. The consumers identify the providers using tag or symbolic name that the ESB interprets, where the tag usually contains the service name and also may contain some additional attributes that influence the routing (priority or policies for example), as is illustrated in the Figure 17 [16].

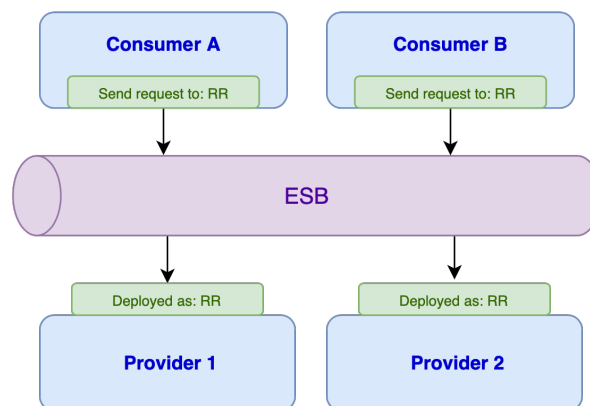


Figure 17: An ESB mediating connections

*Interceptors*

This approach, illustrated in the Figure 18 provides an interceptor or proxy for each provider and for each consumer and the consumers will communicate in a “point-to-point” only with its interceptor [16].

A proxy service is a virtual service that receives messages and optionally processes them before forwarding them to a service at a given endpoint [28].

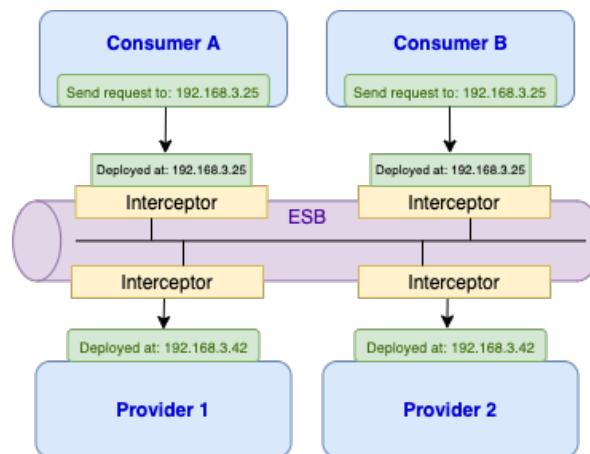


Figure 18: An ESB using interceptors

In this scenario, the process of routing and others ESB services is completely encapsulated from the outside world. In fact, internally, a totally different protocol can be used [16].

*Protocol-Driven Versus API-Driven ESB*

Regarding where the responsibility of an ESB begins from the providers' and consumers' points of view, there are two approaches: protocol-driven and API-driven.

In the protocol-driven approach, the ESB defines a protocol and the providers and consumers send and receive messages according to the chosen protocol, as can be observed in the Figure 19. One example of this approach are the Web Services, which require a SOAP protocol [16].

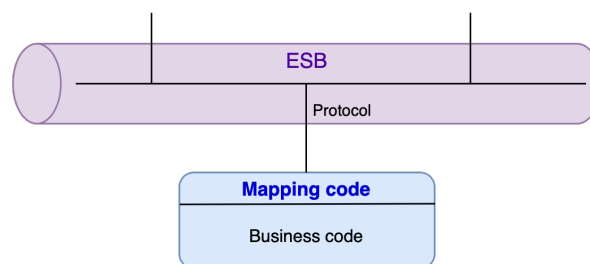


Figure 19: Connecting to a protocol-driven ESB

With the API-driven approach, the ESB defines platform-specific APIs like Java interfaces and the providers and consumers use these for service implementations and service calls (Figure 20).

When the ESB defines only a protocol, the details of how messages are consumed and provided is completely outside the scope of the ESB (and its associated team). It is responsibility of the consumers and providers that use the platform to provide adapters to the protocols and tools used to enable and fulfill the requirements of the protocol [16].

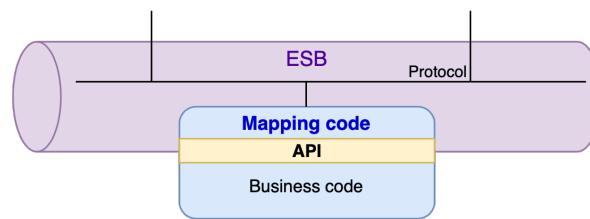


Figure 20: Connecting to an API-driven ESB

When the ESB is responsible for the APIs, the providers and consumers can concentrate on the business functionality. If new platforms get integrated, it is the responsibility of the ESB to provide solutions and maintain the corresponding APIs, providing code generators and libraries [16].

### 2.3.3 Existing solutions

In this Section we present two products for design and implement the ESB: the *WSO2 Enterprise Integrator*<sup>3</sup> and *Mule ESB*<sup>4</sup>.

#### *WSO2 Enterprise Integrator*

The *WSO2 Enterprise Integrator (WSO2 EI)* is a hybrid integration platform that offers runtime integration, multiple deployment options, CLI and monitoring tools. In the center of the integration is the *Micro Integrator (MI)* that can behave like an ESB, a streaming integrator or a microservice. In the rest of this section we will focus what the *Micro Integrator* offers when used as an ESB.

The *Micro Integrator* as an ESB is a lightweight, component oriented and Java based that allows developers to integrate services and applications in a easy, productive and efficient way [6].

It also has a bus architecture that simplify communication between applications, turning software systems with different data formats into a unified architecture. When deployed, it caters to your message routing, transformation, message mediation, service orchestration, as well as service and API hosting needs. So, it acts as a medium carrying data between systems using a message oriented communication mechanism [6].

The following are some considerations about the WSO2 solution:

- Offers a loosely coupled solution with easy to manage plug-in and plug-out of solution components
- Configuration driven design, as opposed to code driven design
- Support multiples transport protocols based on data formats and data storage and destination characteristics including HTTP, HTTPS and JMS

<sup>3</sup> <https://wso2.com/integration/>

<sup>4</sup> <https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb>



- Protects the service from the different protocols and message formats, separating the business logic from the messaging
- Provides multiple tools for data transformation
- Offers a refactorable enterprise by breaking business processes and key competencies into multiple combinations of applications and services that can be reused or combined to create new services or improve products and services rapidly, with proactive patching and regular security updates
- Includes packages with development and debugging tools
- Provides a scalable design suited for enterprise wide deployment
- supports full-cloud, on-premises, hybrid, and multi-cloud deployment without any additional complexity or cost since the runtime and management components can run anywhere within the user's full control.
- Equipped with monitoring and analytic tools to keep track of bus after deployment

### *MuleESB*

Mule ESB is also a lightweight Java-based [ESB](#) and integration platform that allows developers to connect applications together quickly and easily, enabling them to exchange data [\[22\]](#).

As the [WSO2 EI](#), it can also enables integration regardless the type of technology that the application uses, like [JMS](#), Web Services, [JDBC](#) and [HTTP](#). It also can be deployed anywhere and can integrate events in real time or in batch [\[22\]](#).

Some technical features of this solution includes:

- Expose and host reusable services, using the [ESB](#) as a lightweight service container
- Safeguards the services from message formats and protocols and enable location-independent service calls
- Exchange data across the services in multiple formats and transport protocols
- Components can have any type, from [POJOs](#) to components from another framework
- Reusing existing components without any change, since the business logic is kept completely separate from the messaging logic and is it not required any Mule-specific code
- Messages could take any type since is not restricted to SOAP or binary
- Mule does not force any design constraints on the architect such as [XML](#) messaging or [WSDL](#) service contracts.
- Mule's stage event-driven architecture makes it highly scalable

### *WSO2 EI vs Mule ESB*

As we can see from the previous Sections, both solutions offer service mediation, service routing and data transformation as well as a wide range of pre-defined connectors, which are essential for the goal of this dissertation.

One of the differences between them is that while the WSO2 is open source, the open source version of the Mule do not offers the same features as the paid version.

Also, the WSO2 has a pre-designed dashboard for the management and monitoring of the [ESB](#) while the Mule ESB uses the [JMX](#) [21]. Furthermore, the WSO2 also offers a integration Studio, which is a drag-and-drop graphical development environment for [WSO2 EI](#), which provides an efficient integration artifact development and accelerates development lifecycles.

---

## ARCHITECTURE AND WSO2 INTEGRATION

---

The previous Chapters discussed the solutions regarding the integration of multiple systems. Now, we need to apply the studied concepts and knowledge in the definition of our architecture.

Nevertheless, in order to design the most fitting architecture, is necessary to understand the main characteristics and requirements of the applications that may use this solution.

So, this Chapter will focus on the design of a solution based on those characteristics and requirements and present the WSO2 products and components of interest for the implementation of the proposed solution.

### 3.1 ARCHITECTURE

Mobile identification applications are electronic representations of physical cards, documents or a set of attributes that characterizes an individual. We can take as example the mobile drivers license, mobile student card and the Digital Covid Certificate.

These applications need some type of authentication to ensure that the user only receives their data and thus prevent identity theft. The authentication process can be provided by a third-party or by the application itself.

In addition, to provision the applications with the required data, it may need to consult the services where the attributes are stored to retrieve them. However, there may not be a centralized service with all the information. This implies that applications will have to communicate with more than one service (and possibly in different ways) and merge the results.

Finally, retrieved attributes may need to undergo transformations (changing data types or define namespaces). This adds an extra step to the process.

#### *Mobile Identification Applications Requirements*

The mobile identification applications have some characteristics in common, such as:

1. Need to authenticate a user using an internal or third-party service
2. Obtain user attributes by accessing different attribute sources with various types of technologies
3. Normalize the received attributes names with unique identifiers within a namespace.

4. Share data between applications
5. Updated the data and attributes regularly, which implies several communications with both internal and external services.

Considering the characteristics and the requirements of the applications described above, that have a strong impact on the design on the architecture, and the knowledge acquired during the State of the Art, Figure 21 presents the proposed architecture.

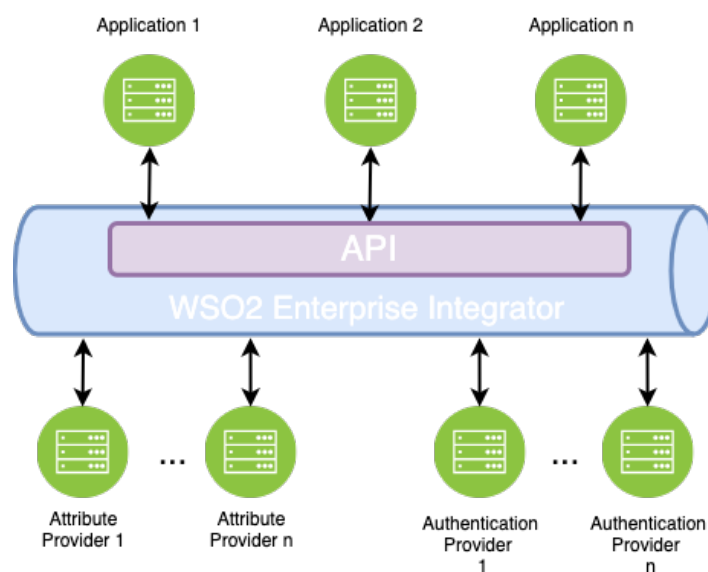


Figure 21: Proposed Architecture

In this architecture there are two types of providers:

- **Authentication Provider** - Entity or service that provides authentication mechanisms to prove the identity of users or system processes
- **Attribute Provider** - Entity or service that provides data and informations related to a specific user

The applications are those who need to use an external authentication provider or access one or more attributes providers to collect information. It is possible to have multiple providers and applications.

Regarding the process flow, the applications interact with the **ESB** through a **REST API**, using the **HTTP** communication protocol. This way, it offers a single interface to communicate. This interface was chosen due to its popularity and it is a simple request/response mechanism, which is suitable for this situation. Also, all the requests and responses will use the **JSON** format it is a universal format widely used by web applications to exchange data.

Then the **ESB** processes the request and communicates with the appropriate provider, adapting to the communication protocol used by it, which removes complexity on the applications side.

Finally, it receives and processes the response from the provider and returns it to the application. The responses attributes must come in the format and with the namespace defined by most applications.

This architecture will be developed using the [WSO2 Enterprise Integrator](#) and the next Chapters will focus on that process.

## 3.2 INTEGRATION WITH WSO2

Founded in 2005, the WSO2 is a middleware vendor that provides a large range of open-source solutions for [API](#) management, integration, analytics, identity and access management, that can be deployed on premises or on the cloud. Furthermore, it also offers solutions specialized for the industry of healthcare, finances and education.

As stated earlier in the Chapter 2, one of their products is the [WSO2 Enterprise Integrator](#) that will be used to implement the case studies in the Chapter 4. In addition, to speed up the development process, the [WSO2 Integration Studio \(WSO2 IS\)](#) will also be used.

### 3.2.1 *WSO2 Enterprise Integrator*

The [WSO2 Enterprise Integrator](#) is a hybrid platform that supports multiple integration architectures styles such as a centralized [ESB](#), microservice or a cloud-native architecture. The main part of the [WSO2 EI](#) is the [MI](#) server, which is an event-driven message engine, which will behave as an [ESB](#) in this solution. It can be deployed on-premises, on the cloud, hybrid or in a container orchestration platform. In addition, the it also provides a dashboard that allows to consult and update the artifacts that are deployed, the logs and manage users.

The latest version at the time this dissertation was written and the one that will be used is the 7.1.0 and can be downloaded from the [WSO2 website](#)<sup>1</sup> by choosing one of the many installation options available. For demonstration purposes, the binary version will be used in this dissertation, as the installation and deploy process is similar to all machines. Inside the home directory "wso2ei-7.1.0", which will be referred throughout this dissertation as **EI\_HOME**, is the Micro Integrator, the Micro Integrator Dashboard and the Streaming Integrator. The last one component is used to capture, analyze, process, and act upon streaming data and events in real-time, will not be used since it doesn't fit the purpose of integrating multiple providers.

### 3.2.2 *WSO2 Integration Studio*

The [WSO2 Integration Studio](#), is a development environment for designing, developing, debugging and testing integration solutions, allowing to accelerate the development process. It can be download from the [WSO2 website](#)<sup>2</sup> and easily installed.

To start developing an integration solution, it is possible to begin with a blank project or with an integration sample that is already configured for the chosen integration scenario. The samples includes content based

---

<sup>1</sup> <https://wso2.com/integration/>

<sup>2</sup> <https://wso2.com/integration/integration-studio/>

routing, database pooling, REST to SOAP conversion and many others. These options are available on the starting page, as seen in Figure 22, along with references to the documentation and integration examples.

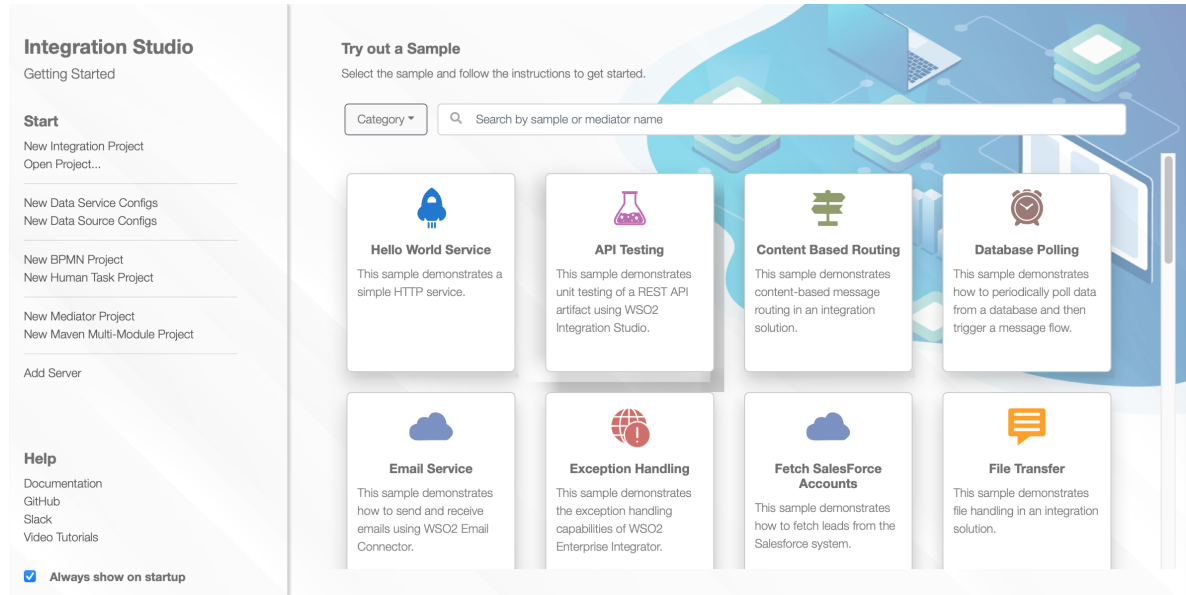


Figure 22: Integration Studio getting started page

The WSO2 IS allows to accelerate the development process by providing multiple tools with specific roles. For implementing an ESB, the main features are:

- **Project Explorer** - List of all the projects directories and respective files in the selected workspace.
- **Editors** - Local where the integration solution is written or edited and can be accessed by creating a REST API, Proxy Service or a Sequence artifact. There are 3 types of editors provided by the WSO2 IS:
  1. **Graphical Editor** - Drag-and-drop tool that provides a pallet of integration mediator to use. Each mediator can be configured according to the needs of the solution.
  2. **Source Editor** - In addition to the graphical editor, it is also possible to write the solution using the source code.
  3. **Swagger Editor** - Available when a REST API is created, it allows to write the solution using a swagger definition.
- **Console** - Displays a variety of logs such as standard output, CVS operation output, errors and stacktrace with hyperlinks to the source code location.
- **HTTP Client** - Embedded RESTful HTTP client to assist in testing.
- **Micro Integrator server** - Embedded Micro Integrator server that allows to easily deploy and test the integration artifacts during the development phase.

- **Runtime Services** - List all the services available after deploy when using the embedded Micro Integrator.

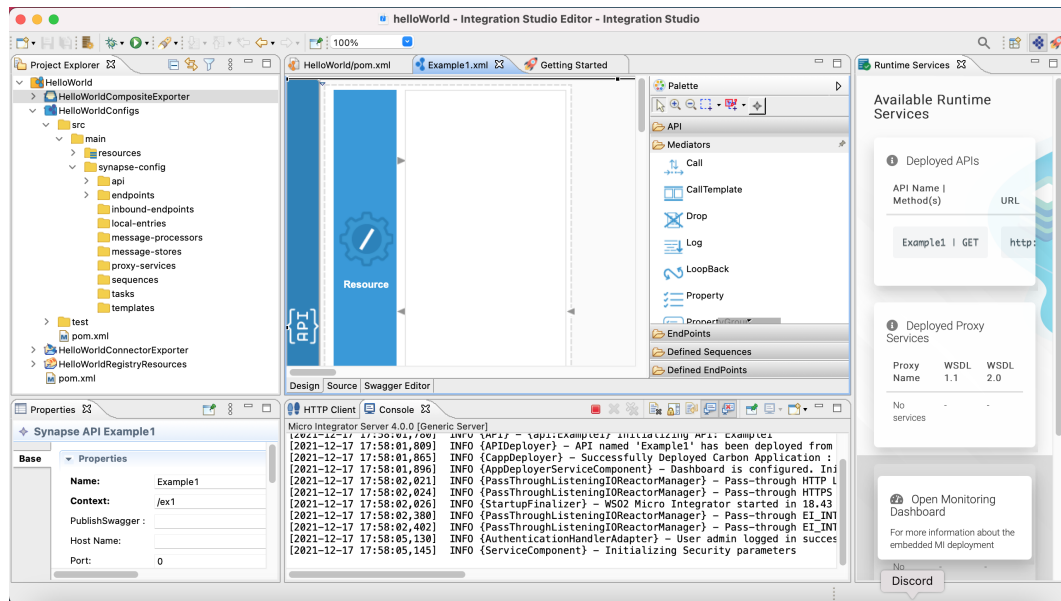


Figure 23: Integration Studio view

### 3.2.3 Integration Project Structure

When creating a project, the WSO2 EI offers multiple integration modules where only the necessary to develop the solution can be included.

The six available modules are: ESB Config, Connector Exporter, Registry Resource, Docker Exporter, Kubernetes Exporter and Composite Exporter. The only required modules for developing an ESB is the ESB Config and the Composite Exporter.

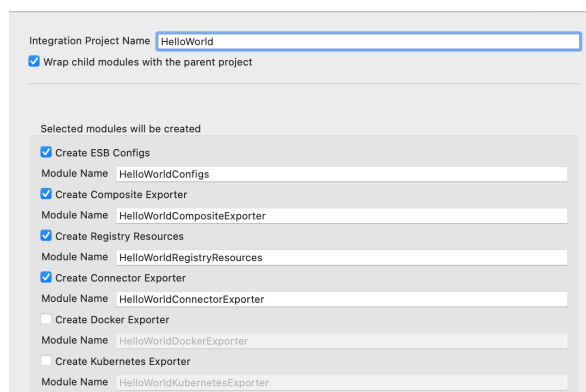


Figure 24: Menu for selecting the modules

### *ESB Config*

The ESB Config is the project directory responsible for store all the ESB artifacts that define the mediation flow. This project includes multiple synapse artifacts that can be added to the integration flow, such as:

- **Proxy** - Virtual web service that receives and processes a message before sending it to a server at a given endpoint. It can be accessed by using a [URL](#) (similar to a normal web service address) and can receive and send messages using the available transport protocols. It is also possible to create a [WSDL](#)-based proxy service.
- **REST API** - Endpoint in the [WSO2 EI](#) that also receives and processes a message before sending it to the respective endpoint. Each [API](#) has a defined [URL](#) context (and will only process requests that fall under that context) and is composed of one or more Resources, which are logical components of an [API](#) that can be accessed by making an [HTTP](#) call. These resources are responsible for mediate incoming requests, forward them to a specified endpoint, mediate the responses from the endpoint, and send the response back to the application that originally requested them.
- **Templates** - Used to avoid code duplication and redundant configurations. The [MI](#) can template sequences and endpoints. These templates may be used later in mediation flow.
- **Sequences** - Set of mediators where the messages are processed and are used in the proxy service and the REST APIs. Sequences are reusable.
- **Scheduled tasks** - Code that will be executed at a specific time.
- **Endpoints** - External service where a message or request must be delivered. May be represented by a [URL](#), mailbox, [TCP](#) socket or [JMS](#) queue.
- **Message Store/Processor** - Used by a mediation sequence to temporarily store messages before they are delivered to their destination. The task of the Message Processor is to pick the messages stored in the Message Store and deliver it to the destination. This approach is useful when dealing with asynchronous message with guarantee that the message was delivered to the respective endpoint.

### *Connector Exporter*

Project directory where the connectors artifacts from the WSO2 connectors store are stored before packaging. These are used in mediation sequence of the [ESB](#) and allows to connect and interact with external services by wrapping the [API](#) of those services. It also provides several connectors for various areas such as communication tools (like Slack [4]), social media tools (such as Twitter [5] or LinkedIn [2]), developers tools (for example GitHub [3]) and many others.



### Registry Resources

The registry is a content store and a metadata repository that stores various configurations and resources like endpoints, XSTL scripts and WSDL. The configuration files are stored using a key-value pair and a file-based registry is configured by default.

### Composite Exporter

Allows to package all the existent artifacts from ESB projects into one composite application (C-APP) which later can be deployed in the Micro Integrator server by generating a CAR file. A composite application orchestrate several independent programs, data or devices to form a solution that they could not provide on their own. [? ]

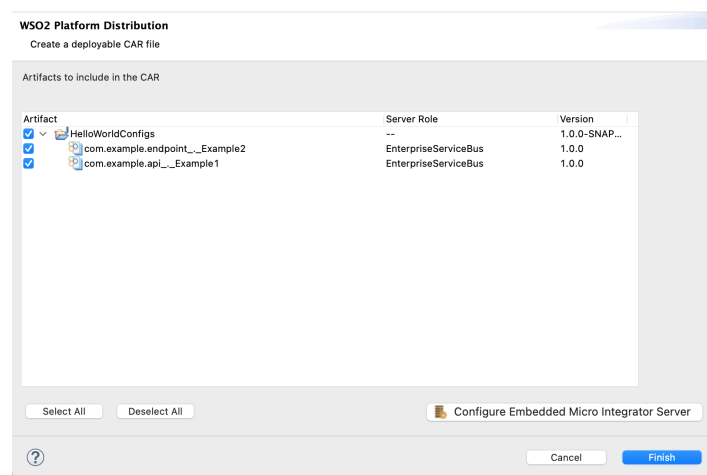


Figure 25: Menu for selecting the artifacts to deploy

### Docker Exporter

This project allows to deploy the integration solution in a docker environment by packing multiple modules and projects into a docker image.

### Kubernetes Exporter

This project allows to deploy the integration solution in a Kubernetes environment by packing multiple modules and projects into a kubernetes image.

3.2.4 ESB Flow

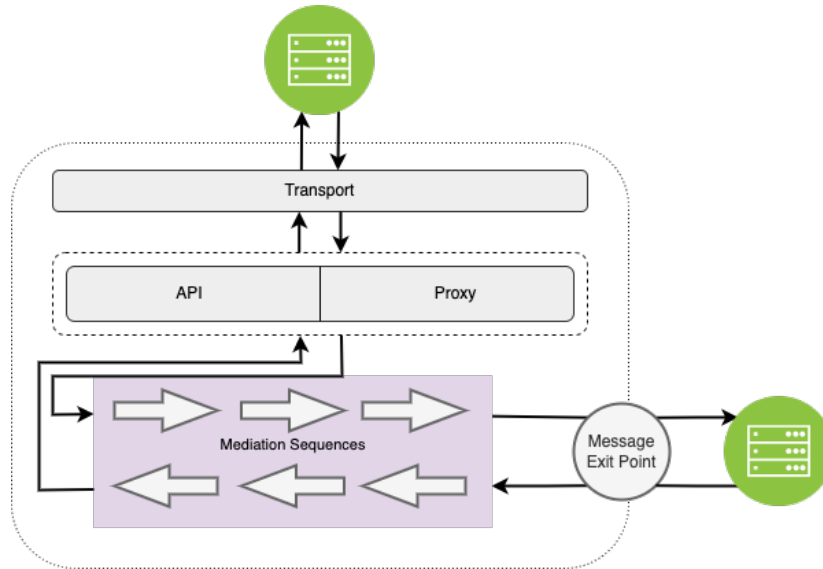


Figure 26: Flow of an ESB-based integration

The Figure 26 illustrates the general flow of an ESB-based integration.

First, the message goes through the transport layer which is responsible for carrying the message in a pre-defined format. The default protocol used is HTTP/HTTPS, but many others such as JMS and VFS are supported. In addition, it is possible to configure each protocol to the project needs.

Then, the message arrives to the Message Entry Point, which are entities through which a message enters the Micro Integrator mediation flow. The entry points may be a REST API or a Proxy Service. Although they have different settings and functions, both define the mediation sequences to process the message.

*Mediation Sequences*

Mediation sequences are a set of mediators organized into a logical flow that allows to perform the necessary message processing and route the message to their destination.

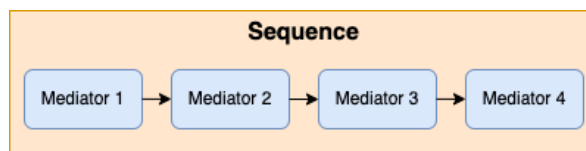


Figure 27: Mediation Sequence

There are 3 main sequences that can be used by the message entry points: the In sequence, the Out sequence, and the default fault sequence.

The **In sequence** is where the message is injected by the message entry point and it is responsible for all the mediation logic from that moment until the message is delivered to its destination.

The **OUT sequence** defines the mediation logic that processes the response message received before sending the it back to the requester.

The **Fault sequence** used for handling errors in the message mediator. When an error occurs in the flow, the message that triggered the error is delegated to the specified fault sequence to receive proper treatment.

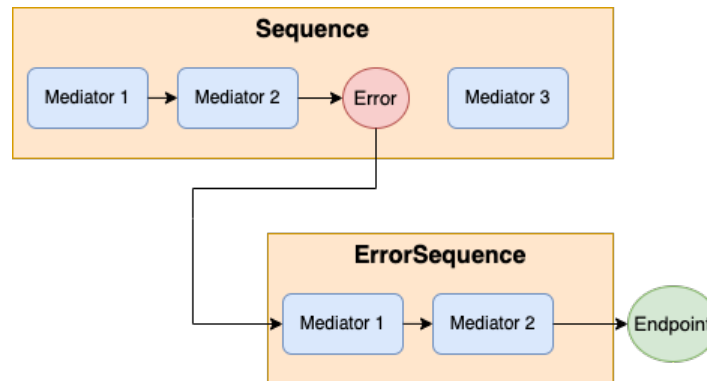


Figure 28: Mediation Sequence after an Error

It is also possible to create named sequences, which are custom and reusable sequences that can be used multiple times in a project. For example, while the In and Out sequences are used to define the flow, the fault sequence is a named sequence because it only combines mediators.

The use of In and Out sequences depends on the number of external services that will be used during the flow. If the message is only delivered to one external service during the entire process, it is recommended to use IN and OUT sequence as described above. However, if it is a service chaining scenario, that is, sending messages to multiple external entities sequentially in one flow, it is recommended to only use the In sequence to implement the whole flow. The chaining scenarios also applies to use situations where the goal is to aggregate multiple named sequences responses.

### *Mediators*

The mediators are individual processing units or actions that perform a specific function on the message. For example, the Call mediator sends the messages to a defined endpoint or connector and the Aggregate mediator collects and merges the multiple responses. The mediators are capable of filtering the message, change its core, transform it, match incompatible protocols and data formats. In addition, is possible to write a costume mediator using Java, Spring and scripting.

The In and Out Sequence use recommendations are connected to the mediators that are responsible for the delivery of the message to the external service: Send and Call. The Send mediator sends the message to an endpoint and the response goes automatically to the OutSequence. The Call mediator sends the message and delivers the response to the next mediator (that is placed right after).

### Message Exit Points

After the message undergoes the necessary changes, it is sent to the external destination also known as message exit point, and it can be an endpoint or a connector.

### 3.2.5 Configuration

#### Micro Integrator Configuration

The Micro Integrator includes TOML-based product configuration, so all server-level configurations can be applied in the **deployment.toml** file, which is located in the *EI\_HOME/micro – integrator/conf* directory.

```
[server]
hostname = "localhost"
# offset = 10

[user_store]
type = "read_only_ldap"

[keystore.primary]
file_name = "repository/resources/security/wso2carbon.jks"
password = "wso2carbon"
alias = "wso2carbon"
key_password = "wso2carbon"

[truststore]
file_name = "repository/resources/security/client-truststore.jks"
password = "wso2carbon"
alias = "symmetric.key.value"
algorithm = "AES"
```

Figure 29: Deployment.toml file snippet

It is possible to configure numerous parameters related to deployment, database connections, system parameters, LDAP user store, API management, Keystones and many others. The complete list of configuration parameters are listed in the [WSO2 documentation](#)<sup>3</sup>.

Also, the Micro Integrator uses the [Synapse](#)<sup>4</sup> as the underlying mediation engine.

The Synapse is a free, open source and complete asynchronous messaging engine. It provides support for XML, REST and Web Services and, for the message interchange format as SOAP, plain text, binary and JSON.

The MI also supports the standard XPath functions and variables through its underlying XPath engine for accessing message properties and uses the syntax to identify and navigate nodes in a XML document.

#### Project Configuration

In the [WSO2 EI](#), the all configuration parameters of the product are defined in multiple files called *pom.xml*.

The *pom.xml* inside the project root contains the project settings and the modules that are included. The main configuration parameters of this file are: the groupId that allows to uniquely identify the project, the artifactId that is the name of the jar without version, the version, the name of the project, the description and identify the

<sup>3</sup> <https://ei.docs.wso2.com/en/latest/micro-integrator/references/config-catalog/>

<sup>4</sup> <https://synapse.apache.org>

modules included in the project. All the parameters must follow the proper name convention from the Apache Maven. For example, a project name may be org.example.integration.sample1.0.0.

In addition, each module added to the project has a configuration file as each of them has different roles and settings.

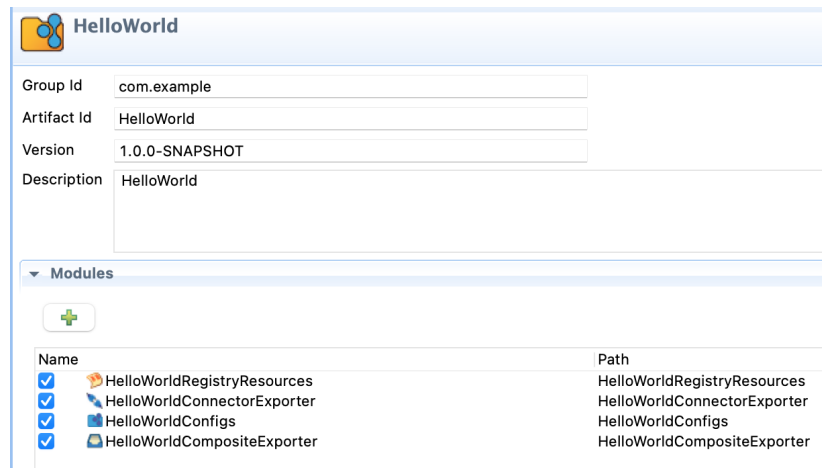


Figure 30: Pom.xml file in the WSO2 IS

### 3.2.6 Build and Run

To build and run the created artifacts, there are 2 options: use [WSO2 Integration Studio](#) or use a local enterprise integrator instance.

#### Using the Integrator Studio

To deploy the artifacts using the embedded Micro Integrator in the [WSO2 IS](#), you just need to go to the Composite Exporter Project, select the option "**Export project Artifacts and Run**" and choose the artifacts that should be included in the build. After the deployment, a console opens with the logs and another with the deployed services. This option is excellent for testing during the development stage because of its speed and ease.

It is also possible to generate a docker image with or without the included the Micro Integrator. This can be achieved by selecting the option "**Generate Docker Image**" after right-click the Connector Exporter project.

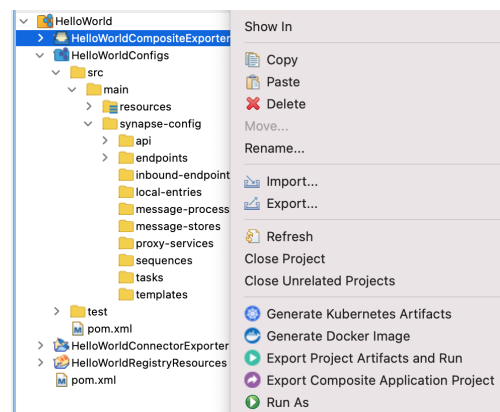


Figure 31: Composite Exporter dropdown

The same process applies for generating the artifacts kubernetes, but selecting the option "**Generate Kubernetes Artifacts**".

These options are excellent for a quick and ease testings during the development stage.

#### *Using a local Micro Integrator instance*

To build and run the artifacts using the local Micro Integrator instance, you must first ensure that the [WSO2 EI](#) is installed on the machine.

Then, export the deployable **CAR** file that contains the artifacts by selecting the option "**Export Composite Application Project**" in the Composite Exporter.

Afterwards, copy the file to the `EI_HOME/micro – integrator/repository/deployment/server/carbonapps` directory.

Finally, start the Micro Integrator by running the command "**sh micro-integrator.sh**" on the `EI_HOME/micro – integrator` directory.

#### *Micro Integrator Dashboard*

After the artifacts are deployed and the Micro Integrator is running, it is possible to access its Dashboard.

If the artifacts were deployed using the embedded Micro Integrator, click Open Monitoring Dashboard in the Runtime Services tab as shown in the [Figure 32](#).

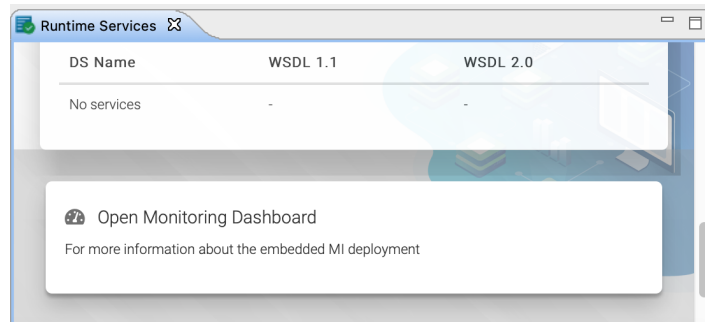


Figure 32: Open Monitoring Dashboard button

If the local instance was used, the dashboard starts by running the command `shdashboard.sh` in the `EI_HOME/micro – integrator – dashboard/bin/directory`.

To access the dashboard, it is necessary to log in using the hostname for the running instance, the port, the username and the password. The admin user and password are defined in the `deployment.toml` file.

The home page of the Micro Integrator dashboard is illustrated in the [Figure 33](#).

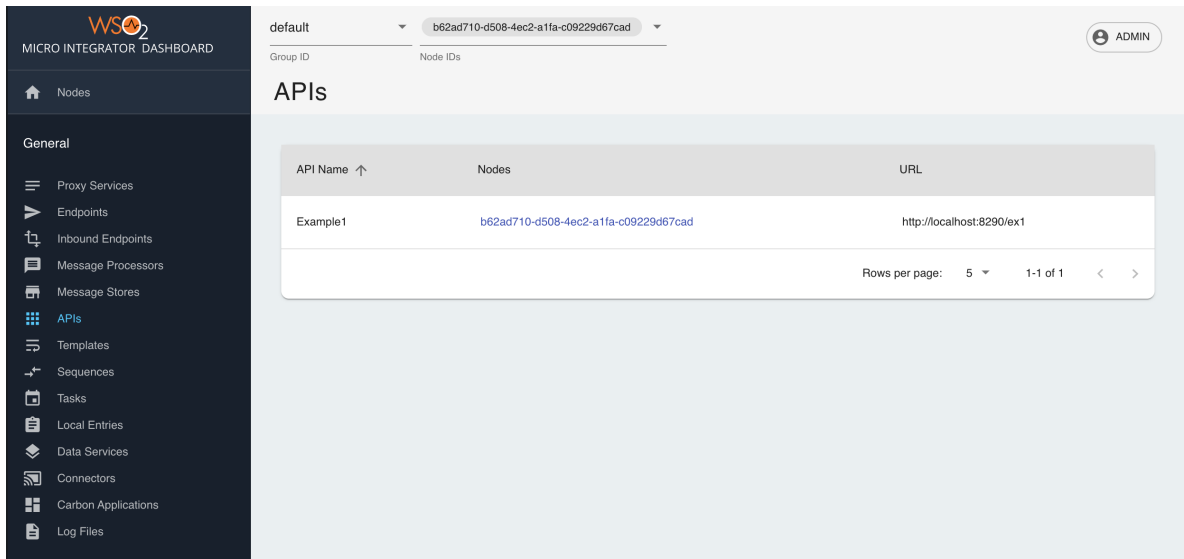


Figure 33: Dashboard view

---

## IMPLEMENTATION

---

As stated in the first Chapter, the goal of this dissertation is to design and implement an architecture capable of integrating external sources of attributes and managing the attributes it receives.

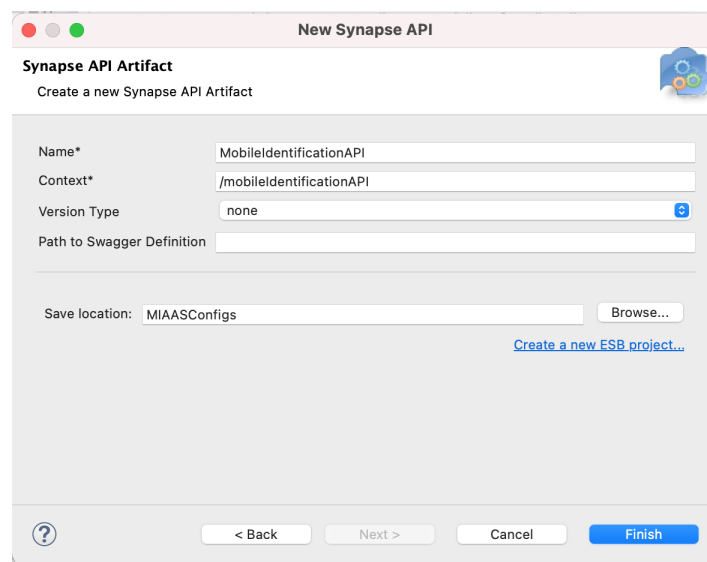
This Chapter will focus on implementing the architecture designed in the Section 3.1 and also implementing 3 use cases to be included on our solution. The three use cases presented are: Authentication using the [Chave Móvel Digital \(CMD\)](#); collection of [IMT](#) attributes; collection of attributes from a [LDAP](#) server.

### 4.1 REST API IMPLEMENTATION

First, a project called [Mobile Identification as a Service \(MIAS\)](#) was created with the following modules: the Composite Exporter, the ESB Config, the Connector Exporter and the Registry Resources.

Afterwards, regarding the project configurations, the Group ID has been updated to `com.mias`.

Then the REST API was created with the following settings as presented in the Figure 34.



The screenshot shows a 'New Synapse API' dialog box. The title bar reads 'New Synapse API'. Below the title bar, it says 'Synapse API Artifact' and 'Create a new Synapse API Artifact'. The dialog contains the following fields and controls:

- Name\*: MobileIdentificationAPI
- Context\*: /mobileidentificationAPI
- Version Type: none (with a dropdown arrow)
- Path to Swagger Definition: (empty text field)
- Save location: MIAASConfigs (with a 'Browse...' button)
- A link: [Create a new ESB project...](#)
- Bottom navigation: ? (help icon), < Back, Next >, Cancel, Finish

Figure 34: REST API creation



The code snippet below presents the first state of the [API](#) after created.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <api context="/mobileIdentificationAPI" name="MobileIdentificationAPI" xmlns="
  http://ws.apache.org/ns/synapse">
3   <resource methods="GET">
4     <inSequence/>
5     <outSequence/>
6     <faultSequence/>
7   </resource>
8 </api>

```

During the entire integration flow, it is possible for errors to occur that prevent the flow to proceed normally. In these situations, an appropriate response must be triggered. In this solution, this response consists of returning an [HTTP](#) response to the application that made the original request with the most suitable [HTTP](#) error code and a more detailed explanation in the [JSON](#) message body. This solution will be used, for example, when the initial request body is not what was expected or if there is an error in the communication with the external entities.

To avoid code duplication, a sequence template was created to implement the fault sequence named Fault-Sequence. This template receives two dynamic parameters that are injected when the template is called in a sequence. The first one is the `errorCode`, which represents the [HTTP](#) status code and the second is the `errorMessage` which describes with more detail the error. Both parameters are mandatory and the default value are 404 for the `errorCode` and "Something went wrong. Please try again" for the `errorMessage`.

```

1 <parameter defaultValue="404" isMandatory="true" name="errorCode"/>
2 <parameter defaultValue="Something went wrong. Please try again" isMandatory="
  true" name="errorMessage"/>

```

Next, the sequence was implemented by setting the [HTTP](#) status code of the response. The mediator property was used, which is conditionally content aware and has impact on the message context. You can alter the context by add, set or remove properties and can retrieve properties at any time using the Synapse [XPath](#) Variables or the `get-property()` function.

The [HTTP](#) status property already exists in the message context and is called `HTTP_SC`, so it was only necessary to replace the default value for the one received in the template. To do so, was used the prefix `$func`, together with the `errorCode` value to define the expression. The variable `$func` is used to refer parameters values that are passed externally to the template.

```

1 <property expression="$func:errorCode" name="HTTP_SC" type="STRING"/>

```

Finally, it was only necessary to define the body of the message by using the PayloadFactory mediator. This mediator transforms or replace messages by configuring the format of the message and mapping the arguments. The arguments can be static or be specified by using an XPath or JSON expression.

```

1 <payloadFactory media-type="json">
2   <format>
3     {
4       "Error": "$1"
5     }
6   </format>
7   <args>
8     <arg evaluator="xml" expression="$func:errorMessage"/>
9   </args>
10 </payloadFactory>

```

In the previous code snippet shows the mediator implementation, the \$1 represents the ErrorMessage argument that will be added to the message dynamically. Once again the \$func is used to access the errorMessage value.

Figure 35 presents a the FaultSequence in a visual way. The source code can be found in the appendix ??

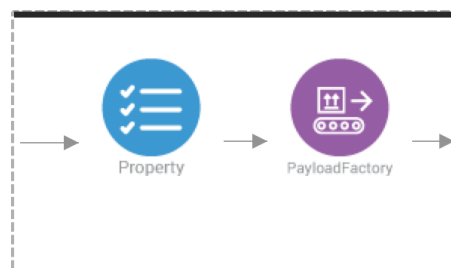


Figure 35: FaultSequence design

## 4.2 AUTHENTICATION USING CHAVE MÓVEL DIGITAL

The CMD is a means of authentication and digital signature certified by the Portuguese State. It allows, in a simple and secure way, to authenticate users on various public and private websites, with just a mobile number and a 4-digit PIN, previously associated with a civil identification number for a Portuguese citizen or with a passport or title/card number of residence for a foreign citizen.

It was developed and is maintained by the *Agência para a Modernização Administrativa (AMA)*, which is the public institute responsible for promoting and developing administrative modernization in Portugal. [1]

The authentication process is divided into two parts: the user authentication flow and the attribute acquisition flow. The website/application where the user wants to log in will be referred as requesting system.

In the first phase, the authentication process is started when the user chooses to authenticate using this method. After authorizing the sharing of personal data with the requesting system (which are previously agreed), the user starts the two-factor authentication by using the mobile number, the PIN and an **OTP** received by message. Phase 1 ends when the requesting system receives a token.

In the second phase, the token is sent to the AMA attribute provider along with the authentication context ID that, if successful, returns the user attributes, thus being able to confirm the user identity.

#### 4.2.1 *Planning*

To implement this use case, the token received by the requesting system at the end of the first phase of authentication is required to start the process of obtaining the user attributes.

First, is checked if the received token in a **JSON** message has the correct format:

```
1 {
2   "token" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
3 }
```

If so, we make an **HTTP POST** request to the attribute provider's **API**, where the message body is a **JSON** that contains the token. If the token is valid, the response is the authentication context ID.

The expected response if the request was successful is:

```
1 {
2   "authenticationContextId" : "xxxxxxxxxxxxxxxxxxxx"
3 }
```

This attribute, together with the token, are needed to make a new request to the **API**, this time using the **GET** method, to receive the available attributes. The token and the authentication context ID are included as parameters in the URL.

The result of this request is a **JSON** with the user attributes, that after is send to the application.

```
1 {
2   [
3     {
4       "name": "NIC_namespace", "value": "123456789"
5     },
6     {
7       "name": "NIF_namespace", "value": "987654321"
8     }
9   ]
10 }
```

### 4.2.2 Implementation

This use case is a typical service orchestration scenario since two sequential [API](#) requests are made. Furthermore, the response from this process can be easily aggregated with another sequence to provide a more complete service. For those reasons, the entire flow was defined in a sequence, so it can be reused in other resources.

Before creating the sequence that implements the integration flow, an endpoint was created for the first [HTTP](#) request. As the first endpoint is responsible for providing the authentication context ID, the endpoint name is **AMA\_AuthenticationContextId**. Its configuration is found in the code snippet below.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <endpoint name="AMA_AuthenticationContextId" xmlns="http://ws.apache.org/ns/
   synapse">
3   <http method="post" statistics="enable" trace="enable" uri-template="http://
   ama.attribute_url.pt">
4     <suspendOnFailure>
5       <initialDuration>-1</initialDuration>
6       <progressionFactor>1.0</progressionFactor>
7     </suspendOnFailure>
8     <markForSuspension>
9       <retriesBeforeSuspension>1</retriesBeforeSuspension>
10    </markForSuspension>
11  </http>
12 </endpoint>
```

Then, the `MobileDigitalKey` sequence was created in order to implement the use case, with the statistics and trace enabled.

The flow begins with the validation of the token received from the request body. If the token was null, the `faultTemplate` sequence was called and the flow ends. If not, the flow proceeded as expected.

This validation started by saving the value of the received token into a property called `token`, using the `json – eval` function to retrieve the value from the JSON message. Then, the filter mediator was used to verify if the token was not null using the [XPath](#), that in this mediator tests the given expression as a Boolean expression.

So, if the token was null, the expression was false and the `FaultSequence` template was called using the parameters `errorCode` with the value 400 and the error Message as "The [JSON](#) message is not valid". Then the response mediator stopped the process of the message and send the response to the application.

If the token was not null, the expression was true and the flow continued to the set of mediators inside the "then" element.

```
1 <filter xpath="get-property('token')">
2   <then>
3     <!--If the expression is true-->
4   </then>
5   <else>
6     <call-template target="FaultTemplate">
7       <with-param name="errorCode" value="400"/>
8       <with-param name="errorMessage" value="The JSON message is invalid"/>
9     </call-template>
10    <respond/>
11  </else>
12 </filter>
```

After verifying that the token was present, the first HTTP request was made to get the authentication context Id, using the call mediator. This mediator invoked the endpoint created at the beginning of the process, the AMA\_AuthenticationContextId.

If the request was successful, that is, the HTTP status code of the response was 200, the flow continued. Otherwise, the FaultSequence template was invoked again with the errorCode as 403 and the errorMessage as "Please enter a valid token" before the respond mediator is used.

```
1 <call>
2   <endpoint key="AMA_AuthenticationContextIdRetriever"/>
3 </call>
4 <filter regex="200" source="$axis2:HTTP_SC">
5   <then>
6     <!--If the expression is true-->
7     <then>
8     <else>
9       <call-template target="FaultTemplate">
10        <with-param name="errorCode" value="403"/>
11        <with-param name="errorMessage" value="Please enter a valid token"/>
12      </call-template>
13      <respond/>
14    </else>
15 </filter>
```

With the token and the authentication context ID values, it was already possible to make a request to the API to obtain the user's attributes. The a new request was made using the call mediator, but this time using an endpoint defined in the sequence.

The endpoint was defined using an [HTTP](#) endpoint with the following settings:

```

1 <endpoint>
2   <http method="get" uri-template="{uri.var.url}">
3     <suspendOnFailure>
4       <initialDuration>-1</initialDuration>
5       <progressionFactor>-1</progressionFactor>
6       <maximumDuration>0</maximumDuration>
7     </suspendOnFailure>
8     <markForSuspension>
9       <retriesBeforeSuspension>1</retriesBeforeSuspension>
10    </markForSuspension>
11  </http>
12 </endpoint>

```

The url-template was defined on the property named uri.var.url, where the token and the authenticationContextID were concatenated to the [URL](#) through the concat function.

```

1 <property expression="json-eval($.authenticationContextId)" name="
  authenticationContextId" scope="default" type="STRING"/>
2 <property expression="fn:concat(fn:concat(fn:concat('https://ama.getAttributes.pt?
  token=', get-property('token')), '&authenticationContextId='), get-
  property('authenticationContextId'))" name="uri.var.url" scope="default" type=
  "STRING"/>

```

At last, if the request was successful, the response from the endpoint was send directly to the application that made the original request. If not, once again the FaultSequence template was invoked again with the errorCode as 502 and the errorMessage as "An unexpected error occurred, please try again".

The source code for this use case can be found in the [Section A.3](#) in the Appendices.

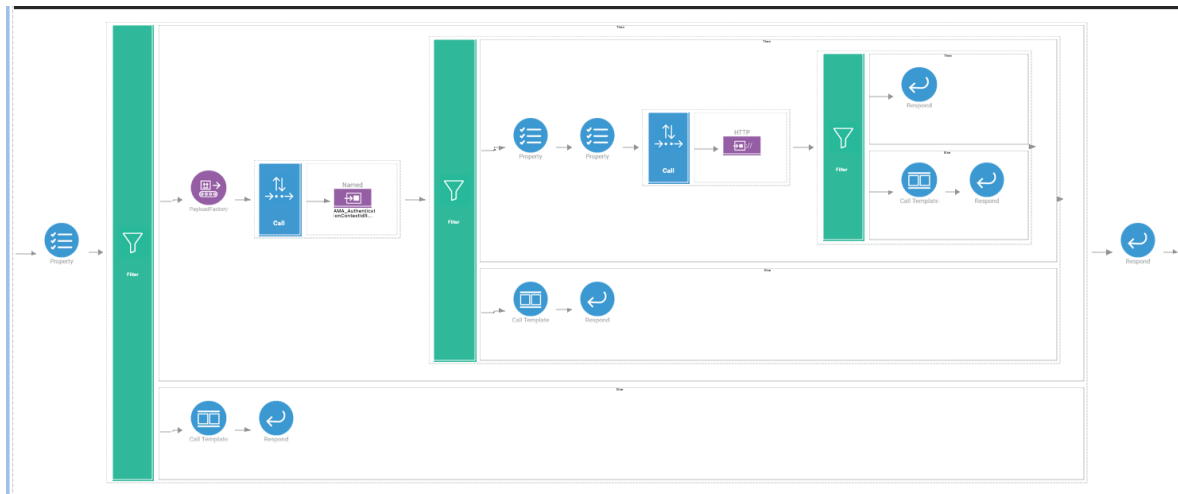


Figure 36: Visual implementation of the CMD use case

### 4.3 ATTRIBUTES FROM IMT

The **IMT**, is a public institute integrated in the indirect administration of the State, endowed with administrative and financial autonomy and its own assets. Its mission is to regulate, supervise and exercise coordination and planning functions in the land transport sector, including the qualification of drivers, certification of vehicles and railway infrastructure, and the supervision and regulation of this sector.

This case study will focus on the role of the **IMT** as an entity that manages driving licenses. As such, they have all the information relating to a user.

The purpose of this case study is to obtain attributes by querying its services.

#### 4.3.1 Planning

There will be 2 phases to implement this use case: the first is to retrieve the attributes from the **IMT** service and the second is to manage them.

For retrieving the attributes, the **IMT** provides a **SOAP** based web service and a **WSDL** file that describes the location of the service, the operations that can be performed, the required elements, the protocol and data format for each operation. The **WSDL** will be referred as **WSDL\_IMT** for the rest of the Section.

For the **SOAP** request, the applications needs to supply the **Tax Identification Number (NIF)**, the **Civil Identification Number (NIC)** and the birth date in the original request. If the **JSON** message contains this attributes, the request to the **IMT** web service is made.

If the request is successful, the final step is to manage the received attributes. This process has 3 purposes:

1. Transform the message type from **SOAP** to **JSON**
2. Remove the unnecessary attributes

### 3. Change the attributes namespaces

The first reason is due to all responses from our API are in the JSON format. The second and third are to make the attributes compliant with the ISO/IEC 18013-5:2021<sup>1</sup>.

This ISO defines the Mobile Driver License (mDL) application, including the namespaces of all the mandatory and optional attributes that compose a drivers license. Therefore, making the attributes ISO compliant reduces the complexity of the application.

At last, the result of this transformation is sent to the application.

#### 4.3.2 Implementation

Although this is not a service orchestration scenario, this use case could be aggregated with other sequences. For this reason the implementation was made in a sequence called IMTSequence.

The first mediator used in the sequence was the validate mediator. Since there were 3 parameters in the JSON message to verify, this mediator validates the request message by comparing it with the JSON schema that represented the expected message. The schema was:

```

1 {
2   "NIC" : "123456789",
3   "NIF" : "987654321",
4   "birthDate" : "1988-02-02"
5 }
```

If the validation failed, the FaultSequence template was invoked inside the on-fail tag and the mediation flow ends with the respond mediator. If not, the flow continues.

```

1 {
2   <validate [source="xpath"]>
3     <property name="validation-input-messa" value="true"/>
4     <schema key="driverLicenseSchema"/>
5     <on-fail>
6       <call-template target="FaultTemplate">
7         <with-param name="errorCode" value="400"/>
8         <with-param name="errorMessage" value="The JSON message is invalid"/>
9       </call-template>
10      <respond/>
11    </on-fail>
12  </validate>
13 }
```

<sup>1</sup> <https://www.iso.org/standard/69084.html>



Before sending the request to the **IMT** web service, the **SOAP** message was created using the PayloadFactory mediator. The Soap Envelope was defined according to the information provided in the WSDL\_IMT file and the attributes received in the original request were injected the envelope as arguments.

Then, the envelope was sent to the web service using the call mediator and a **WSDL** endpoint as the target endpoint. This type of endpoint extracts the target endpoint reference from a **WSDL** document specified in the configuration. To configure the endpoint is required the name of the **WSDL** file (that could be a file or a **URL**), the service name and the port.

```

1 <call>
2   <endpoint>
3     <wsdl>
4       <suspendOnFailure>
5         <initialDuration>-1</initialDuration>
6         <progressionFactor>-1</progressionFactor>
7         <maximumDuration>0</maximumDuration>
8       </suspendOnFailure>
9       <markForSuspension>
10        <retriesBeforeSuspension>0</retriesBeforeSuspension>
11      </markForSuspension>
12    </wsdl>
13  </endpoint>
14 </call>

```

If the response from the web service contained the attributes, the operation was successful. However, if any error occurs while invoking the external service, the faultSequence template was called again and the flow ended.

The final stage of this flow was to manage the received attributes by using Data Mapper Mediator.

The Data Mapper Mediator allows to convert and transform one data format to another and change the structure of the data in a message by using a Data Mapper engine. The data transformation may be arithmetic, Boolean, type conversion or conditional. The engine also provides a real time Data Mapper Preview for testing the mapping solution.

The data mapper configuration includes the input and output message type (**JSON**, **XML**, **CVS**) and the input and output schema files. This files represents the format of the message that the mediator is expected to receive and transform to, respectively. All the configurations are saved in the project registry, including the schema files.

The input schema in this use case is the soap envelope received from the **IMT** database and the output is a **JSON** message. The output already excludes the attributes that are not relevant and maps the attributes. For example, the value of the dataNascimento in the input message was the value of the birthDate in the output message. Some attributes needed to undergo some transformation, such as being transformed from a string to a integer.

The Figure 37 illustrate the graphic Mapping Configuration file generate for this use case. The input and output schema file can be found in the Section A.4 in the Appendices.

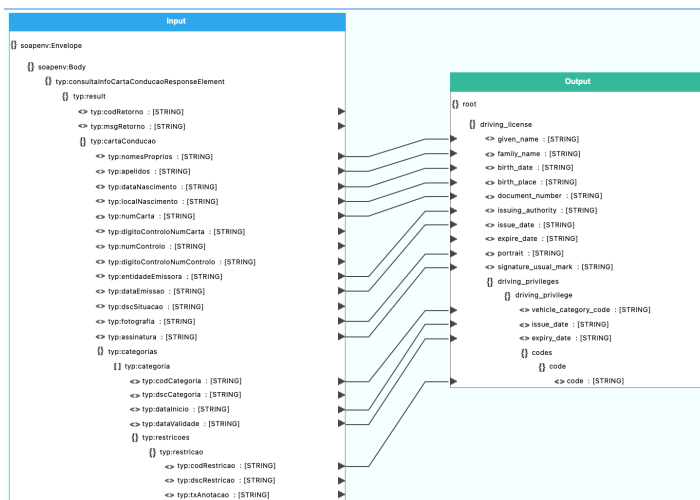


Figure 37: Mapping

Then, the JSON message create is delivered to the application and the flow ends. The source code for this use case can be found in the Section A.4 in the Appendices.

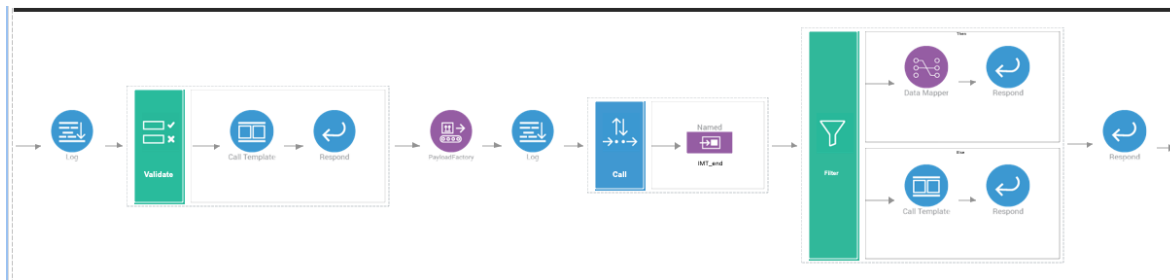


Figure 38: Visual implementation of the IMT use case

#### 4.4 LDAP

The LDAP is a flexible and well supported standard-based mechanism for interacting with directory servers, which are central repositories that store and manage general-purpose data. It is often used for authentication and storing information about users, groups, and applications.

This protocol allows clients to perform multiple operations in a directory server, such as authentication and information storage about users, groups, and applications, data retrieval and search data using filters.

In this use case, the LDAP will be used to obtain the attributes of students from a university a LDAP server, which are named LDAP entries.

Each student has a full name, date of birth, course and year and a student number, which will be the unique identifier.

#### 4.4.1 Planning

To implement this use case we will use one of the connectors provided by the WSO2: The **LDAP** connector. This connector allows to connect to any **LDAP** server through a web service interface and create, read, update and delete operation on their entries. In addition, it uses the **JAVA JNDI APIs**<sup>2</sup> to connect to the server.

First, the connector must be downloaded from the **WSO2 Connector Store**<sup>3</sup> and added to the project.

This connector offers 2 API resources to communicate with the server: the create and the search. The first one creates a new **LDAP** entry while the second performs a search on one or more entities with the specified search keys.

Before performing any operation, our solution must be authenticated in order to obtain access to the server. For this process is necessary:

- `providerUrl` - The URL of the **LDAP** server
- `securityPrincipal` - The **DN** of one admin of the server.
- `securityCredentials` - The password of the **LDAP** admin.
- `secureConnection` - The boolean value for the secure connection.
- `disableSSLCertificateChecking` - The boolean value to check whether certificate enable or not.

After the access is granted, a request is made to the search resource on the find the student, using the student number as a filter.

Finally, the response from the connector is sent to the applications.

#### 4.4.2 Implementation

Once again, this use case was implemented in a sequence called **LDAPSequence** to be reused with other sequences, providing a more complex response.

Before starting implementing the sequence, the **LDAP** connector was downloaded and the zip file was added to the **ConnectorExporter** by right-clicking on the directory and selecting the option "Add/Remove Connector".

The first mediator added to the **LDAPSequence** was the filter mediator, to verify if the body of the request contained the student number. If it was not present, the **FaultSequence** template was invoked to stop the flow and inform the applications that the **JSON** message is not valid. Otherwise, the flow continued.

The next step is to authenticate with the **LDAP** server in order to obtain access to perform the **LDAP** operation. To do so, the `<ldap.init>` element was added to the sequence. The values were defined in a property.

```
1 <ldap.init>
2   <providerUrl>{$ctx:providerUrl}</providerUrl>
```

2 <https://directory.apache.org/api/user-guide.html>

3 <https://store.wso2.com/store/assets/esbconnector/ldap>

```

3 <securityPrincipal>{$ctx:securityPrincipal}</securityPrincipal>
4 <securityCredentials>{$ctx:securityCredentials}</securityCredentials>
5 <secureConnection>{$ctx:secureConnection}</secureConnection>
6 <disableSSLCertificateChecking>{$ctx:disableSSLCertificateChecking}</
  disableSSLCertificateChecking>
7 </ldap.init>

```

After the access is granted, the operation for searching a given student is created, using the student number as a filter is:

```

1 <ldap.searchEntry>
2 <limit>1000</limit>
3 <filters>{json-eval("studentNumber")}</filters>
4 <dn>x</dn>
5 <attributes/>
6 </ldap.searchEntry>
7 <respond/>

```

The response from the [LDAP](#) server was sent to the application.

The source code of this use can be found on the [Section A.5](#) in the Appendices.

## 4.5 IMPLEMENTING THE SOLUTION

After the use cases were implemented, they were included in the [API](#) created at the beginning of this Chapter.

As stated earlier, each use case is a resource in the [API](#), with an [URL](#) context, an [HTTP](#) method and a In sequence.

```

1 <resource inSequence="CMD" methods="POST" url-mapping="/authentication/CMD" />
2 <resource inSequence="IMT" methods="POST" url-mapping="/attributes/IMT"/>
3 <resource inSequence="LDAP" methods="POST" url-mapping="/attributes/StudentLDAP"/
  >

```

In the code snippet above are the definition of the three resources. All resources uses the [HTTP](#) POST method, since each one requires data from the applications in order to provide the service and because the mediation flow was defined using one sequence, the outSequence parameter is not required.

The context of the sequences are defined using an [URL](#) mapping that include the provider type and its name. This way is easier to identify if it is an authentication provider or an attribute provider. For example, the [CMD](#) is an authentication provider, so the URL-mapping is "authentication/CMD". For the InSequence, its value is the name of the sequence created in the previous Subsections that corresponds to the resource context.

Lastly, was also created the required default resource, which does not specify an [URL](#) mapping or template and is invoked when a request do not match any of the definitions on the other resources. It sends a [JSON](#) response to the application to inform that an error has occurred.

```
1 <resource methods="DELETE POST PUT GET">
2   <inSequence>
3     <call-template target="FaultTemplate">
4       <with-param name="errorCode" value="404"/>
5       <with-param name="errorMessage" value="Resource not found"/>
6     </call-template>
7   </inSequence>
8 </resource>
```

The implementation of the [API](#) and the respective resources can be found in the [Section A.1](#) of the Appendice.

This solution is very versatile as it works as a catalog of reusable services for multiple applications. For instance the authentication resource can be used to authenticate an user in citizen card application or in the vaccination certificate application.

Moreover, a new resource could be easily created by aggregating existent sequences into a new sequence. Take as example the [CMD](#) sequence and the [IMT](#) sequence. In the new sequence, first the [CMD](#) sequence could be used to obtain the all the available attributes, including the ones necessary to perform the request to the [IMT](#) server. Then the request would be made and the responses from both request would be aggregated and sent to the application.

The next Chapter will focus on how this solution can be integrated into an existing project: the [Mobile Driver License](#).

---

## INTEGRATION WITH MOBILE ID

---

This Chapter will demonstrate how the solution implemented in the previous Chapter can be integrated with Mobile Identity applications. In order to achieve this goal, the Mobile Driving License project will be used.

The purpose of the **Mobile Driver License (mDL)** is to represent the physical driving license and facilitate the exchange of information between the holder of the license and other entities such as the police.

The **mDL** is composed of 3 major components: the **mDL holder app** which is the application where the license holder can view his driving license and choose what to share with other entities; the **mDL reader**, which is the application used by entities to obtain a driving license or just some attributes, such as name, age, image, among others; and the **issuing authority**, which manages all the applications data and is the communication point between the applications and external entities. Figure 39 illustrates the components and their exposed interfaces, based on the ISO/IEC 18013- 5:2021.

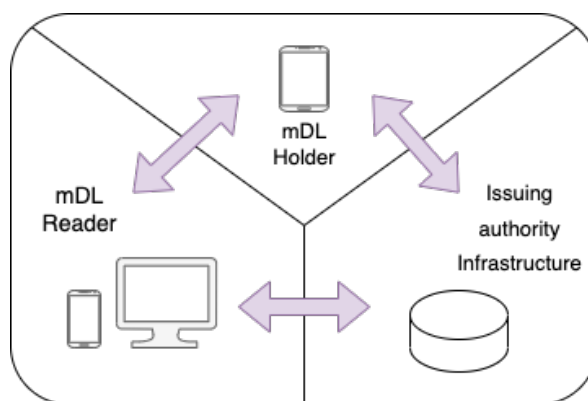


Figure 39: mDL components

There are 4 phases in the **mDL** lifecycle that ensure the correct use of transactions between applications and within them:

- Registry - In this phase the user identifies himself by providing his data. That data is validated and saved on the issuing authority
- Emission: emission phase of the driving license

- Usage: corresponds to all processes inherent to the use of the driving license during the various transactions;
- Management: corresponds to the set of processes involved in managing the user attributes and identity in the system.

In this proof of concept we are going to focus only on the registration and management phase.

In the registration, the first step is for the user to authenticate himself. At this moment, the authentication is performed by using a two-factor authentication service created in the backend service called Simulated Source, with the purpose to recreate the **CMD** authentication or using the **CMD** itself.

Finally, after it is proven that the user is who she/he claims, the backend queries its database to get the user attributes. If the result is empty, it uses the **IMT** service to obtain the attributes, transform them and store them in the database. The entire communication process with external entities and the transformation of attributes is performed by a backend service called **mDL Association**.

Regarding the maintenance phase, the data needs to be frequently updated to guarantee the veracity of the information. As such, periodic requests are made to external services that provide the attributes through a service called **mDL Refresh**.

Figure 40 present in more detail the communication made by the issuing authority without the proposed solution. In this version, the requests are made directly to the external attribute source by the modules.

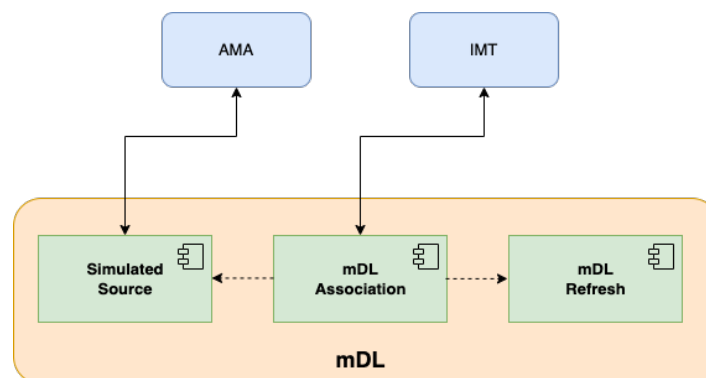


Figure 40: Issuing authority architecture before using the proposed solution

With the introduction of the proposed architecture, the communications to external services are now mediated by the **API**, which is the entry to the **ESB** as illustrated in the Figure 41.

Moreover, since the data transformations are done by the **ESB**, the complexity and required work made by the issuing authority reduces. As a example, the **IMT** resource transforms the attributes received from the endpoint according to the nomenclature and types defined by the **ISO**.

This proves that our solution can be used by any type of application without altering its behavior. It also demonstrate that multiple resources can be added to the **API** without affecting the applications that consumes it. In addition, it removes the complexity from the **mDL** of maintaining the integrations, as any adjustment that need to be performed are made in the proposed architecture and are invisible to the applications.

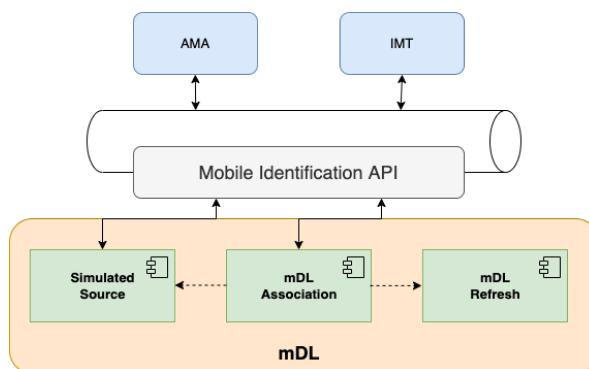


Figure 41: Issuing authority architecture after using the proposed solution

These principles can be apply to all Mobile Identity applications.



---

## CONCLUSION

---

### 6.1 FINAL CONSIDERATIONS

The purpose of this dissertation was to design and implement a architecture that was able to integrate external services in runtime and manage the system without creating dependencies between the mobile identity applications and the solution.

So, naturally, the first step was to study the state of the art in this matter, including the existing solutions. So, it started by understanding the concept of the [Service Integration and Management \(SIAM\)](#) and how this methodology could be applied in an environment with multiple providers to ease the regulation of the interactions between them. To do so, was presented in detail its key concepts, ecosystem and structures available.

Then, since one of the methodologies presented was similar to a [Service-Oriented Architecture \(SOA\)](#), the concepts, characteristics and patterns of this type of architecture were presented. However, it was also listed the challenges of implementing a [SOA](#).

After, we focused on one of the main components of the [SOA](#), the [Enterprise Service Bus \(ESB\)](#), that allowed multiple services to communicate with each other, and introducing its concepts, responsibilities and possible designs.

Finally, was presented two existing solutions: the Mule ESB and the [WSO2 Enterprise Integrator \(WSO2 EI\)](#). After performing an analysis of both solutions, the [WSO2 EI](#) was chosen because all features are open-source, unlike Mule ESB, it has all the features we want for this project and provided the [WSO2 IS](#) tool, which facilitates and accelerates solution development process.

The next step was create our architecture. We started by studying what kind of applications would benefit from it and its main characteristics and requirements and from there we design our proposal.

Then, the components of the [WSO2 EI](#) and the [WSO2 IS](#) that were used to implement our solution were presented, such as the REST API that served as an entry point for the [ESB](#), the sequences of mediators that implemented the flow, the connectors that facilitate the integration with multiple well known services and all the required configurations.

Afterwards, the proposed architecture was implemented and included 3 use cases: Authentication using the [Chave Móvel Digital](#), the collection of [IMT](#) attributes and the collection of attributes from a [LDAP](#) server.

At last, a proof of concept was presented by applying the proposed architecture to an existing mobile identity application, the [Mobile Driver License](#). As a result of this integration, was proved that including the proposed solution into the application would not change its behaviour, but would reduce the complexity, the work load and the dependencies with external service providers. It also allowed the application to use the available services during its lifecycle. Finally, the results of this proof of concept were translated to all the Mobile Identification applications.

## 6.2 FUTURE WORK

As the digital transformation advances, is natural that new mobile identity applications emerge with more and new requirements than those already discussed throughout this dissertation. For this reason, it will be necessary to integrate more service providers into our [API](#), in order to provide the applications the most complete service's catalog.

Other aspect that could be improved is the format of the request and responses handled by the [API](#). At this moment it only accepts [JSON](#) as the format on the payload, but others, such as [XML](#), could be added to offer more options to the applications.

At last, as was explained in the previous Chapter, the mobile applications have their own internal services, like the mock authentication service. Since it is a useful service, it could be added to our [API](#) to be used by all the applications that utilize our [API](#).

---

## BIBLIOGRAPHY

---

- [1] O que é a chave móvel digital? URL <https://www.autenticacao.gov.pt/a-chave-movel-digital>.
- [2] LinkedIn. URL <https://www.linkedin.com>.
- [3] Github. URL <https://github.com>.
- [4] Slack. URL <http://slack.com>.
- [5] Twitter. URL <https://twitter.com>.
- [6] Samisa Abeysinghe. *What is WSO2 ESB?* WSO2, 2017.
- [7] Atos. Service integration and management (siam) implementation : the benefits and challenges. URL <https://atos.net/wp-content/uploads/2016/06/atos-siam-implementation-brochure.pdf>.
- [8] Martin Breest. An introduction to the enterprise service bus. URL [https://www.cin.ufpe.br/~in1062/intranet/bibliografia/fose-the\\_enterprise\\_service\\_bus.pdf](https://www.cin.ufpe.br/~in1062/intranet/bibliografia/fose-the_enterprise_service_bus.pdf).
- [9] David A. Chappell. *Enterprise Service Bus: Theory in Practice*. "O'Reilly, 2004.
- [10] Featured customers. Featured customers that trust soa. URL <https://www.featuredcustomers.com/vendor/soa/customers>.
- [11] IBM Cloud Education. *ESB (Enterprise Service Bus)*. IBM, 2019.
- [12] IBM Cloud Education. *SOA (Service-Oriented Architecture)*. IBM, 2019.
- [13] Ruben Franzen. *What is SIAM: Service integration and management?* 2020.
- [14] GSMA. Mobile identity enabling the digital world. Technical report, 2020.
- [15] Jeremy H. *What Is an ESB and Its Alternatives?* DreamFactory, 2020.
- [16] Nicloai M. Josuttis. *SOA in Practice*. O'Reilly Media, Inc, first edition edition, 2007.
- [17] Moran Lev. *The Key Whys and Whats of SIAM*. ITSM Tools, 2017.
- [18] Martin Luenendonk. *How to Build a Service Oriented Architecture (SOA)*. Cleverism, 2019.

- [19] Michelle Major-Goldsmith. *A SIAM Trilogy Part 1: Considering a move to a SIAM model*. Scopism.
- [20] MuleSoft. What is an esb?, . URL <https://www.mulesoft.com/resources/esb/what-esb>.
- [21] MuleSoft. Jmx management, . URL <https://docs.mulesoft.com/mule-runtime/3.9/jmx-management>.
- [22] MuleSoft. *What is Mule ESB?* MuleSoft, 2021.
- [23] Hannah Patterson. *The Impact of Cisco ServiceGrid in aHyperdistributed World*. Cisco, 2015.
- [24] Juniper Research. Digital identity: Technology evolution, regulatory analysis forecasts 2019-2024. Technical report, 2019.
- [25] Scopism. *Service Integration and Management (SIAM) Foundation Body of Knowledge*. Scopism, 2017.
- [26] Scopism. Service integration; a different service management art form, 2019. URL <https://www.scopism.com/service-integration-a-different-service-management-art-form/>. Online.
- [27] André Vieira. *What Is a Service-Oriented Architecture?* Outsystems, 2020.
- [28] WSO2. *Working with Proxy Services*. WSO2, 2020.

Part I

APPENDICES



---

## WSO2 ENTERPRISE INTEGRATOR SOURCE CODE

---

### A.1 MOBILEIDENTIFICATIONAPI

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <api context="/mobileIdentificationAPI" name="MobileIdentificationAPI" xmlns="
  http://ws.apache.org/ns/synapse">
3   <resource methods="DELETE POST PUT GET">
4     <inSequence>
5       <call-template target="FaultTemplate">
6         <with-param name="errorCode" value="404"/>
7         <with-param name="errorMessage" value="Resource not found"/>
8       </call-template>
9       <respond/>
10    </inSequence>
11    <outSequence/>
12    <faultSequence/>
13  </resource>
14  <resource inSequence="AMA" methods="POST" url-mapping="/authentication/CMD">
15    <outSequence/>
16    <faultSequence/>
17  </resource>
18  <resource inSequence="Attributes" methods="POST" url-mapping="/attributes/IMT
  ">
19    <outSequence/>
20    <faultSequence/>
21  </resource>
22  <resource inSequence="LDAP" methods="POST" url-mapping="/attributes/
  LDAPUniversity">
23    <outSequence/>
24    <faultSequence/>
25  </resource>
26 </api>
```

## A.2 FAULTSEQUENCE TEMPLATE

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <template name="FaultSequence" xmlns="http://ws.apache.org/ns/synapse">
3   <parameter defaultValue="407" isMandatory="false" name="errorCode"/>
4   <parameter defaultValue="Something went wrong. Please try again" isMandatory=
5     "false" name="errorMessage"/>
6   <sequence>
7     <property expression="$func:errorCode" name="HTTP_SC" type="STRING"/>
8     <payloadFactory media-type="json">
9       <format>
10        {
11          "Error": "$1"
12        }
13      </format>
14      <args>
15        <arg evaluator="xml" expression="$func:errorMessage"/>
16      </args>
17    </payloadFactory>
18  </sequence>
19 </template>

```

## A.3 CHAVE MÓVEL DIGITAL RESOURCE

## A.3.1 Chave Móvel Digital Sequence

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <sequence name="CMD" statistics="enable" trace="enable" xmlns="http://ws.apache.
3   org/ns/synapse">
4   <property expression="json-eval ($.token)" name="token" scope="default" type="
5     STRING"/>
6   <filter xpath="get-property('token')">
7     <then>
8       <payloadFactory media-type="json">
9         <format>
10          {
11            token: "$1"
12          }
13        </format>
14        <args>
15          <arg evaluator="xml" expression="get-property('token')"/>
16        </args>

```

```

15     </payloadFactory>
16     <call>
17         <endpoint key="AMA_AuthenticationContextIdRetriever"/>
18     </call>
19     <filter regex="200" source="$axis2:HTTP_SC">
20         <then>
21             <property expression="json-eval($.authenticationContextId)"
22                 name="authenticationContextId" scope="default" type="
23                 STRING"/>
24             <property expression="fn:concat(fn:concat(fn:concat('https:/
25                 ama.getAttributes.pt?token=', get-property('token')), '&
26                 amp;authenticationContextId='), get-property('
27                 authenticationContextId'))" name="uri.var.url" scope="
28                 default" type="STRING"/>
29             <call>
30                 <endpoint>
31                     <http method="get" uri-template="{uri.var.url}">
32                         <suspendOnFailure>
33                             <initialDuration>-1</initialDuration>
34                             <progressionFactor>-1</progressionFactor>
35                             <maximumDuration>0</maximumDuration>
36                         </suspendOnFailure>
37                         <markForSuspension>
38                             <retriesBeforeSuspension>1</
39                             retriesBeforeSuspension>
40                         </markForSuspension>
41                     </http>
42                 </endpoint>
43             </call>
44             <filter regex="200" source="$axis2:HTTP_SC">
45                 <then>
46                     <respond/>
47                 </then>
48                 <else>
49                     <call-template target="FaultTemplate">
50                         <with-param name="errorCode" value="502"/>
51                         <with-param name="errorMessage" value="An
52                             unexpected error occurred, please try again"/>
53                     </call-template>
54                     <respond/>
55                 </else>
56             </filter>
57         </then>
58         <else>
59             <call-template target="FaultTemplate">
60                 <with-param name="errorCode" value="403"/>

```



```

53         <with-param name="errorMessage" value="Please enter a
54             valid token"/>
55     </call-template>
56     <respond/>
57 </else>
58 </filter>
59 </then>
60 <else>
61     <call-template target="FaultTemplate">
62         <with-param name="errorCode" value="400"/>
63         <with-param name="errorMessage" value="The JSON message is not
64             valid"/>
65     </call-template>
66     <respond/>
67 </else>
68 </filter>
69 </respond/>
70 </sequence>

```

### A.3.2 AMA Endpoint

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <endpoint name="AMA_AuthenticationContextId" xmlns="http://ws.apache.org/ns/
3     synapse">
4     <http method="post" statistics="enable" trace="enable" uri-template="http://
5         ama.attribute_url.pt">
6         <suspendOnFailure>
7             <initialDuration>-1</initialDuration>
8             <progressionFactor>1.0</progressionFactor>
9         </suspendOnFailure>
10        <markForSuspension>
11            <retriesBeforeSuspension>0</retriesBeforeSuspension>
12        </markForSuspension>

```

## A.4 IMT RESOURCE

## A.4.1 IMT Sequence

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <sequence name="imt" trace="disable" xmlns="http://ws.apache.org/ns/synapse">
3   <validate cache-schema="true">
4     <schema key="IMT_payload_schema"/>
5     <on-fail>
6       <call-template target="FaultTemplate">
7         <with-param name="errorCode" value="400"/>
8         <with-param name="errorMessage" value="The JSON message is not
          valid"/>
9       </call-template>
10      <respond/>
11    </on-fail>
12  </validate>
13  <payloadFactory media-type="xml">
14    <format>
15      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
          envelope/">
16        <soapenv:Header/>
17        <soapenv:Body>
18          <typ:consultaInfoCartaConducaoElement xmlns:typ="http://carta
          /types/">
19            <typ:input>
20              <typ:idSistema>x</typ:idSistema>
21              <typ:nif>
22                <typ:codPaisNIF>"x"</typ:codPaisNIF>
23                <typ:numNIF>$1</typ:numNIF>
24              </typ:nif>
25              <typ:docIdentificacao>
26                <typ:numDocumento>$2</typ:numDocumento>
27              </typ:docIdentificacao>
28              <typ:dataNascimento>$3</typ:dataNascimento>
29              <typ:f1ObterImagensExterno>"x"</
                typ:f1ObterImagensExterno>
30            </typ:input>
31          </typ:consultaInfoCartaConducaoElement>
32        </soapenv:Body>
33      </soapenv:Envelope>
34    </format>
35    <args>
36      <arg evaluator="xml" expression="get-property('NIF')"/>
37      <arg evaluator="xml" expression="get-property('NIC')"/>
38      <arg evaluator="xml" expression="get-property('birthDate')"/>

```

```

39     </args>
40 </payloadFactory>
41 <call>
42     <endpoint key="IMT_end"/>
43 </call>
44 <filter xpath="//yp:CartaConducaoResponseElement/typ:codRetorno/text()">
45     <then>
46         <datamapper config="gov:datamapper/IMTMapper.dmc" inputSchema="
            gov:datamapper/IMTMapper_inputSchema.json" inputType="XML"
            outputSchema="gov:datamapper/IMTMapper_outputSchema.json"
            outputType="JSON" xsltStyleSheet="gov:datamapper/
            IMTMapper_xsltStyleSheet.xml"/>
47     <respond/>
48 </then>
49 <else>
50     <call-template target="FaultTemplate">
51         <with-param name="errorCode" value="502"/>
52         <with-param name="errorMessage" value="An unexpected error
            occurred when contacting the endpoint. Please try again"/>
53     </call-template>
54     <respond/>
55 </else>
56 </filter>
57 </sequence>

```

### IMT WSDL Endpoint

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <endpoint name="IMT_end" xmlns="http://ws.apache.org/ns/synapse">
3     <wsdl port="IMTPort" service="CartaConducao" statistics="enable" trace="
        enable" uri="file:/IMT.wsdl">
4         <suspendOnFailure>
5             <initialDuration>-1</initialDuration>
6             <progressionFactor>1.0</progressionFactor>
7         </suspendOnFailure>
8         <markForSuspension>
9             <retriesBeforeSuspension>1</retriesBeforeSuspension>
10        </markForSuspension>
11    </wsdl>
12 </endpoint>

```

## A.4.2 IMT Data Mapping

## IMT Data Mapper Input File

```

1
2
3 <?xml version='1.0' encoding='utf-8'?>
4 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:tp="http://carta/tp/">
5   <soapenv:Body>
6     <tp:CartaConducaoResponseElement>
7       <tp:result>
8         <tp:codRetorno?></tp:codRetorno>
9         <tp:msgRetorno?></tp:msgRetorno>
10        <tp:cartaConducao>
11          <tp:nomesProprios?></tp:nomesProprios>
12          <tp:apelidos?></tp:apelidos>
13          <tp:dataNascimento?></tp:dataNascimento>
14          <tp:localNascimento?></tp:localNascimento>
15          <tp:numCarta?></tp:numCarta>
16          <tp:digitoControloNumCarta?></tp:digitoControloNumCarta>
17          <tp:numControlo?></tp:numControlo>
18          <tp:digitoControloNumControlo?></tp:digitoControloNumControlo>
19          <tp:entidadeEmissora?></tp:entidadeEmissora>
20          <tp:dataEmissao?></tp:dataEmissao>
21          <tp:dscSituacao?></tp:dscSituacao>
22          <tp:fotografia>cid:1025617693662</tp:fotografia>
23          <tp:assinatura>cid:1457609265456</tp:assinatura>
24          <tp:categorias>
25            <!--Zero or more repetitions:-->
26            <tp:categoria>
27              <tp:codCategoria?></tp:codCategoria>
28              <tp:dscCategoria?></tp:dscCategoria>
29              <tp:dataInicio?></tp:dataInicio>
30              <tp:dataValidade?></tp:dataValidade>
31              <tp:restricoes>
32                <!--Zero or more repetitions:-->
33                <tp:restricao>
34                  <tp:codRestricao?></tp:codRestricao>
35                  <tp:dscRestricao?></tp:dscRestricao>
36                  <tp:txAnotacao?></tp:txAnotacao>
37                </tp:restricao>
38                <tp:restricao>
39                  <tp:codRestricao?></tp:codRestricao>
40                  <tp:dscRestricao?></tp:dscRestricao>
41                  <tp:txAnotacao?></tp:txAnotacao>
42                </tp:restricao>

```

```

43         </tp:restricoes>
44     </tp:categoria>
45     <tp:categoria>
46         <tp:codCategoria>?</tp:codCategoria>
47         <tp:dscCategoria>?</tp:dscCategoria>
48         <tp:dataInicio>?</tp:dataInicio>
49         <tp:dataValidade>?</tp:dataValidade>
50         <tp:restricoes>
51             <!--Zero or more repetitions:-->
52             <tp:restricao>
53                 <tp:codRestricao>?</tp:codRestricao>
54                 <tp:dscRestricao>?</tp:dscRestricao>
55                 <tp:txAnotacao>?</tp:txAnotacao>
56             </tp:restricao>
57         </tp:restricoes>
58     </tp:categoria>
59 </tp:categorias>
60 </tp:cartaConducao>
61 </tp:result>
62 </tpCartaConducaoResponseElement>
63 </soapenv:Body>
64 </soapenv:Envelope>

```

### IMT Data Mapper Output Schema

```

1 {
2   "driving_license":{
3     "given_name":"given_name",
4     "family_name":"family_name",
5     "birth_date":"1998-05-01",
6     "birth_place":"birth_place",
7     "document_number":"document_number",
8     "issuing_authority":"issuing_authority",
9     "issue_date":"2021-05-03",
10    "expire_date":"1998-05-03",
11    "portrait":"portrait",
12    "signature_usual_mark":"signature_usual_mark",
13    "driving_privileges": {
14      "driving_privilege":{
15        "vehicle_category_code":"vehicle_category_code",
16        "issue_date":"1998-05-01",
17        "expiry_date":"1998-05-01",
18        "codes":{
19          "code":{

```

```

20         "code": "code"
21     }
22 }
23 },
24 "driving_privilege": {
25     "vehicle_category_code": "vehicle_category_code",
26     "issue_date": "1998-05-01",
27     "expiry_date": "1998-05-01",
28     "codes": {
29         "code": {
30             "code": "code"
31         }
32     }
33 }
34 }
35 }
36 }

```

## A.5 LDAPUNIVERSITY RESOURCE

```

1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <sequence name="LDAP" trace="disable" xmlns="http://ws.apache.org/ns/synapse">
4     <filter xpath="json">
5         <then>
6             <propertyGroup>
7                 <property value="true" name="secureConnection" scope="default"
8                     type="STRING"/>
9                 <property value="true" name="disableSSLCertificateChecking" scope="
10                    default" type="STRING"/>
11                <property value="http://university.ldap.com" name="providerUrl" scope="
12                    default" type="STRING"/>
13                <property value="admin" name="securityPrincipal" scope="default" type="
14                    STRING"/>
15                <property value="admin" name="securityCredentials" scope="default" type="
16                    STRING"/>
17            </propertyGroup>
18            <ldap.init>
19                <providerUrl>{$ctx:providerUrl}</providerUrl>
20                <securityPrincipal>{$ctx:securityPrincipal}</securityPrincipal>
21                <securityCredentials>{$ctx:securityCredentials}</securityCredentials>
22                <secureConnection>{$ctx:secureConnection}</secureConnection>

```

```
18     <disableSSLCertificateChecking>{$ctx:disableSSLCertificateChecking}</  
    disableSSLCertificateChecking>  
19 </ldap.init>  
20     <ldap.searchEntry>  
21     <limit>1000</limit>  
22     <filters>{json-eval("studentNumber")}</filters>  
23     <dn>x</dn>  
24     <attributes/>  
25 </ldap.searchEntry>  
26 </then>  
27 <else>  
28     <call-template target="FaultTemplate"/>  
29     <respond/>  
30 </else>  
31 </filter>  
32 </sequence>
```

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

Este trabalho é financiado por fundos nacionais através da FCT – Fundação para a Ciência e a Tecnologia, I.P., no âmbito do projeto UIDB/50014/2020.