Deliverable D1.3

EMERALD solution architecture-v1

Editor(s):	Iñaki Etxaniz
Responsible Partner:	TECNALIA Research & Innovation
Status-Version:	Final-v1.0
Date:	31.10.2024
Type:	R
Distribution level (SEN, PU):	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	D1.3 EMERALD solution architecture-v1
Due Date of Delivery to the EC	31.10.2024

Workpackage responsible for the Deliverable:	WP1 - Concept and methodology of EMERALD
Editor(s):	Iñaki Etxaniz (TECNALIA)
Contributor(s):	FABA, TECNALIA, Fraunhofer, CNR, SCCH
Reviewer(s):	Christian Banse (Fraunhofer) Cristina Martínez, Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP3, WP4, WP5

Abstract:	Initial version of the description and design of the architecture of the EMERALD solution and underlying component integration.
Keyword List:	Architecture, Requirements, Sequence diagrams, Component cards
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 4.0 International (**CC BY-SA 4.0 DEED** https://creativecommons.org/licenses/by-sa/4.0/)
Disclaimer	Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	14.06.2024	Table of Contents, Structure	Iñaki Etxaniz (TECNALIA)
v0.2	24.09.2024	First draft. Included requirements, Sequence diagrams	Iñaki Etxaniz (TECNALIA)
v0.3	01.10.2024	Completed context and architecture	Iñaki Etxaniz (TECNALIA)
v0.4	15.10.2024	Included 3.4 Analysis, Conclusions Ready for internal review	Iñaki Etxaniz (TECNALIA)
v0.5	24.10.2024	Internal QA Review	Christian Banse (Fraunhofer)
v0.6	25.10.2024	Addressed comments received in the Internal QA review	Iñaki Etxaniz (TECNALIA)
v0.7	30.10.2024	Final review	Cristina Martínez /Juncal Alonso (TECNALIA)
v0.8	31.10.2024	Address comments received in the final review	Iñaki Etxaniz (TECNALIA)TECNALIA
v1.0	31.10.2024	Submitted to the European Commission	Cristina Martínez /Juncal Alonso (TECNALIA)

Table of contents

List of tables

List of figures

Terms and Abbreviations

AI	Artificial Intelligence
AI-SEC	AI Security Evidence Collector
AIC4	AI Cloud Service Compliance Criteria Catalogue
AMOE	Assessment and Management of Organizational Evidence
API	Application Programming Interface
BDR	Business-Driven Requirement
CaaS	Certification-as-a-Service
CI/CD	Continuous Integration / Continuous Delivery
CKM	Cryptography and Key Management
CLI	Command Line Interface
CSA or EU CSA	EU Cybersecurity Act
CSP	Cloud Service Provider
CSV	Comma-Separated Values
CPU	Central Processing Unit
DoA	Description of the Action
EBSI	European Blockchain Services Infrastructure
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
ICT	Information Communications Technology
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JPA	Java Persistence API
KPI	Key Performance Indicator
KR	Key Result
MARI	Mapping Assistant for Regulations with Intelligence
ML	Machine Learning
MS	MileStone
MVC	Model, View, Controller
NFR	Non-Functional Requirement
NLP	Natural Language Processing
OSCAL	Open Security Controls Assessment Language
OSS	Open-Source Software
Protobuf	Protocol Buffers
RBAC	Role-Based Access Control
RCM	Repository of Controls and Metrics
REST	Representational State Transfer
SARIF	Static Analysis Results Interchange Format
SDLC	Software Development Life Cycle
SSI	Self-Sovereign Identity System
TWS	Trustworthiness System
UI/UX	User Interface / User Experience
UML	Unified Modelling Language
VM	Virtual Machine
UI/UX	User Interface/ User eXperience

Executive Summary

This deliverable proposes an architecture for the EMERALD framework. It is produced in the context of *WP1-Concept and methodology of EMERALD,* more concretely in *Task 1.2 EMERALD architecture*. It provides a general view of the EMERALD framework, which complements the Data Model presented some months before in D1.1 [1]. This document contributes to these outcomes of the work package:

- The architecture of the overall EMERALD software suite and the related structural and behavioural models, as well as data modelling and interaction mechanisms definition.
- The integration of WP2, WP3 and WP4 outcomes in the EMERALD audit suite.
- The methods to support the integration of pilots in WP5.

This document is divided in three main parts. The first part presents an overview of the EMERALD framework. A context diagram has been included, showing the main inputs, outputs, and roles involved in the EMERALD workflow. Twelve different components of EMERALD are presented, as well as and the interaction among them. A Glossary of terms closes this part, where the definition of terms helps to understand the EMERALD context.

The second part of the document presents the requirements elicited for the EMERALD framework. The requirements elicitation is an iterative process, mixing several perspectives, where Technical requirements (functional and non-functional), User Interface requirements and Pilot requirements are gathered independently. Afterwards, they are linked, integrated and analysed. We present the tools used to implement the process: GitLab Issues as the requirement definition and tracking tool; Component Cards template to describe components and PlantUML to the create the UML diagrams.

Then, we describe the technical requirements elicited in the first 12 months of the project, grouped by components. They cover the expected functionalities of EMERALD framework. These are complemented by non-functional requirements, that cover a range of properties like performance, security, deployment, or availability, to cite some. These are system constrains which are transversal to many (or all) components. The pilot requirements, worked in WP5, are listed too, and then a mapping with the technical requirements has been presented.

Next, an analysis of the requirements set has been performed, studying their relations, status, and coverage. For that, a set of traceability matrices shows the alignment of the elicited requirements with respect to the EMERALD Key Results, and which technical requirements implement a pilot requirement. To end, a prioritization matrix reflects which requirements will be implemented in each iteration of the EMERALD workplan.

The last part of the document presents the EMERALD Framework detailed view, where each component is described in detail -functionality, interfaces, and behavioural model- using the previously mentioned artifacts. The general data model is also included.

Future version of this document is D1.4 [2], due at M24. It will provide and actualized set of requirements and their status, as design development tasks evolve. The next related task is the integration of the v1 version of the components into the first version of the integrated EMERALD framework, which will be produced in M18 of the project and reported in D1.7 [3].

1 Introduction

1.1 About this deliverable

This deliverable is the result of Task 1.2 – EMERALD architecture, in the WP1-Concept and methodology of EMERALD. Its main goal is to provide a common definition of the EMERALD Framework.

The document includes an overview first, and a detailed description later of the EMERALD architecture. It describes the different components, modules, interactions and interfaces. A concise view of each component is presented, using a template named "Component Card", which contains key information about the component, such as: functionality, interfaces, sub-parts, and license. The component behaviour description is completed by UML sequence diagrams[1], that show the interaction with the rest of components.

The document provides a complete list of the technical requirements of the EMERALD CaaS framework. Part of them have been gathered and developed in cooperation with WP5 - that deals with the pilots' implementation - and WP4 - which oversees the user experience and interaction in the EMERALD framework. Most of the requirements listed here have been already described in more detail in the deliverables of WP2 and WP3 (dedicated to describing the components in depth), WP4 (related to the UI) and WP5 (related to the pilots). An analysis of the requirements, their prioritization and status are also included.

During the first year of the project, several workshops have been conducted among the work packages to coordinate the different views that stakeholders could have about what the EMERALD framework has to provide and how. One of the outcomes are the requirements gathered here.

1.2 Document structure

The remainder of the document is organized as follows:

Section 2 presents a global view of the EMERALD framework, its users and context. The section also includes a Glossary that captures the main terminology used in the project.

Section 3 outlines the methodology and tools used in requirement management and documentation. The functional and non-functional requirements of the EMERALD Framework are presented, along with their priority and current status of implementation. A dashboard finalizes the section.

Section 4 describes the architecture of the EMERALD CaaS framework. It provides a succinct description of the components that make up the EMERALD framework, their workflows, implemented interfaces, and sequence diagrams.

Section 5 presents the conclusions, a summary of findings and outcomes.

Finally, *APPENDIX A: Current status of requirements* contains the list of Technical requirements and their current fulfilment status.

[1] https://en.wikipedia.org/wiki/Sequence_diagram

2 Overview of the EMERALD Framework

This section contains the context diagram of EMERALD and the involved roles, introduces the framework, and provides a Glossary of the most relevant terms used in EMERALD.

2.1 Context diagram

The context diagram of a system shows the roles involved, the basic workflow, as well as the inputs and outputs of the process.

The roles that take part in the EMERALD ecosystem, as well as personas and scenarios, are being investigated in the workshops related to tasks T4.1 – Requirements engineering with compliance managers and auditors and T4.2 – Modelling work processes, in WP4. Table 1 summarizes the main roles in EMERALD. For more information on this subject, consult the deliverables D4.1 [4] and D4.3 [5].

Table 1. Roles in THE EMERALD ecosystem

Generic Role	Roles	Description
Compliance Stakeholders	**Compliance Manager**	Supports the company in being trustworthy, overseeing audit processes, being up to date with security standards, organizing audits and managing the scheduling of different compliance schemes. Creates an audit scope in EMERALD to manage the certification process.
	Compliance Manager for financial services	Focuses on risk management of third-party cloud services, assesses controls based on risk and regulation, manages contractual agreements, and monitors compliance
	Metric Owner	Their tasks consist of defining metrics, collecting evidence for controls and assigning and delegating control implementation to **Technical Implementers**. NOTE: alternatively called **Internal Control Owner**
Auditor Stakeholders	**Internal Auditor**	Reviews all controls of an audit scope. If some are non-compliant, checks the reasons and informs the Compliance Manager.
	External Lead Auditor	In charge of managing the audit process, planning, reporting, and maintaining contact with customers. NOTE: both Auditors are a unique role in the EMERALD UI.
	External Technical Auditor	
Technical Stakeholder	**Technical Implementer**	Performs the technical tasks to implement an assigned control, through software development, configuration, etc. Selects a set of metrics that matches the controls, implements them, and informs the Metric Owner. NOTE: alternatively called **Metric Implementor**

A first categorization divides the roles in three groups according to their function: (i) the **Compliance Stakeholders** (Compliance Managers and Metric Owner) that manage the certification process, organizing audits and preparing the system; (ii) the **Auditor Stakeholders** (Internal and External Auditors), that deal with the results of the assessment of an Audit Scope and report the result to the Compliance Manager; (iii) the **Technical Stakeholder**, who implements the required metrics for the Control owner.

A second categorization can be established among roles that are external to the company being certificated (External Auditors) and the roles internal to the company (the rest of them). The technical implementer is a special case. In fact, they are EMERALD developers, regardless of whether they are internal or external to the company.

Figure 1 depicts a context diagram of the EMERALD framework. It shows the roles involved in the certification workflow, as well as the inputs and outputs of the process.

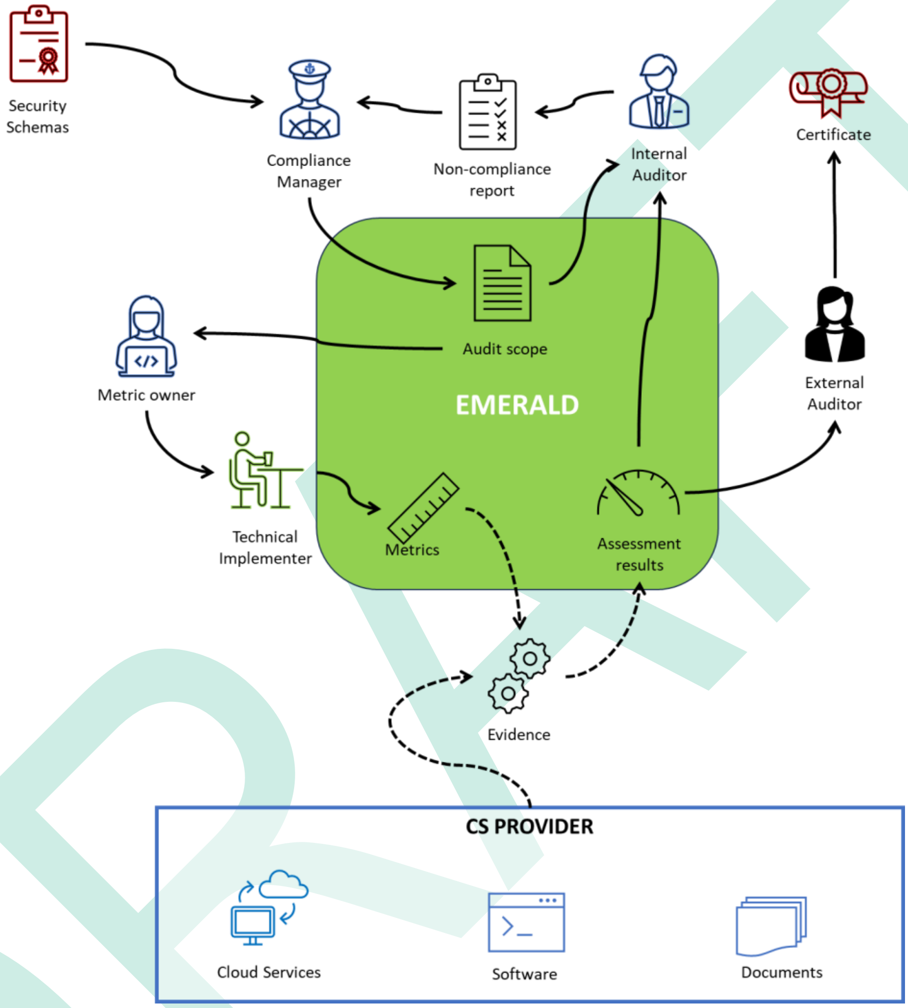


Figure 1. EMERALD context diagram

The main input is the Security Schema, which is used by the **Compliance Manager** to define an Audit Scope. Other inputs are the Evidence, that are gathered by the EMERALD evidence extractors from the Cloud Service Provider, more specifically from the cloud services, the documentation and the software artifacts that the CSP provides (we also call this Certification Target).

The **Control/Metric Owner** assigns the implementation of needed metrics to the **Metric Implementer**. These Metrics will be part of the extractors implementation, and will contribute to provide Evidence and, subsequently, Assessment Results. These are the base for the **Internal Auditor** to produce a Non-compliance Report, and for the **External Auditor** to decide about the Certification granting.

The main output of the process is the mentioned Certificate, that ensures the compliance of the audited services with the Security Schema. The certification is actually produced by a Certification Body, on whose behalf the **External Auditor** works.

2.2 The EMERALD framework

Figure 2 shows a view of the principal EMERALD components and the general data flow between them, as defined in D1.1 [1]. The lines indicate connections between the components, with the arrows indicating the direction of the information flow. The components are coloured according to the respective work package they are related to. The colour also classifies the component regarding it function in the framework (which is also associated with the work package where the component is developed).

There are two types of lines in the diagram. Both indicate flow of data among two components, but in a different mode:

- dashed line (- ->): when a component calls and pulls data from the other component using his API.
- full line (→): when a component actively pushes data to another component using its API.

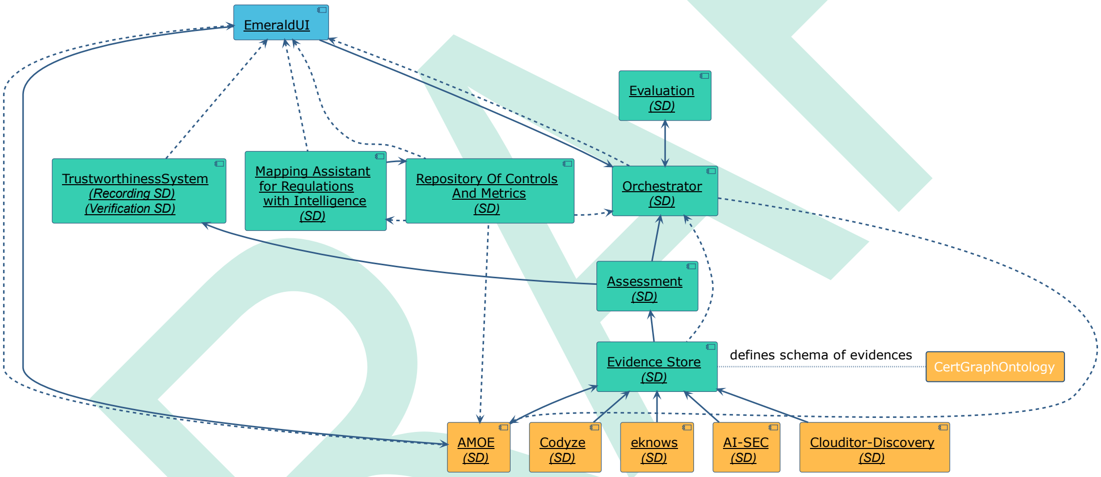

EmeraldUI
Evaluation (SD)
TrustworthinessSystem (Recording SD) (Verification SD)
Mapping Assistant for Regulations with Intelligence (SD)
Repository Of Controls And Metrics (SD)
Orchestrator (SD)
Assessment (SD)
Evidence Store (SD)
defines schema of evidences
CertGraphOntology
AMOE (SD)
Codyze (SD)
eknows (SD)
AI-SEC (SD)
Clouditor-Discovery (SD)

Figure 2. Overview of the EMERALD Components

From bottom to top, the diagram shows the different components of EMERALD framework.

Evidence collectors (in orange) collect different forms of data and extract evidence that are then shared in the EMERALD framework:

- ***AI-SEC*** is an evidence collection tool that extracts various security and robustness information from AI models.
- ***AMOE – Assessment and Management of Organisational Evidence*** – extracts evidence from policy PDF documents. The component stores the uploaded files, as well as relevant metadata related to the document and metrics.
- ***Clouditor-Discovery*** is an evidence gathering tool which extracts Cloud configurations for different Cloud resources (e.g., Virtual Machines, Storage, Networks) from different Cloud providers via API calls.
- ***Codyze*** is a static source code analysis tool which analyses source code of applications comprising Cloud services and assesses security-relevant implementation details according to specified security requirements.

- ***eknows*** is a tool that extracts evidence from source code files collected from the Cloud Service environment, using multi-language reverse engineering.

Evidence assessment and certification components (in green) are the next step in the EMERALD workflow:

- The ***Evidence Store*** functions as a centralized repository for storing evidence from the evidence collector components during the certification process. It utilizes a graph-based database to organize and manage evidence in an efficient and accessible manner.
- The ***Assessment*** component is responsible for assessing the evidence and providing the Orchestrator with assessment results. It calculates the assessment results using the metrics provided by the Repository of Controls and Metrics (RCM).
- The ***Orchestrator****'s* main purpose is to hold all dynamic information about the current audit process, such as the Certification Target, Assessment Results and the Certificate state. It includes the certification graph, providing a snapshot of the cloud service's state.
- The ***Evaluation*** component is responsible for combining assessment results of individual metrics relevant to a specific control of a certification scheme to create an evaluation result for this control.
- The ***Repository of Controls and Metrics (RCM)*** component serves as a smart catalogue of controls and metrics. The repository can contain different schemes, with the corresponding categorization. It also provides import/export mechanisms to facilitate the reuse and composition of catalogue elements.
- The ***Mapping Assistant for Regulations with Intelligence (MARI)*** component is an intelligent system using AI techniques and NLP processing to select suitable metrics for demonstrating compliance with certification schemes. It can also associate security controls of two different certification schemes.
- The **Trustworthiness System (TWS)** component ensures that all actions and data within the certification process are tamper-proof and verifiable. It securely stores the information and associated metadata of evidence and assessment results on a general-purpose Blockchain network.

Finally, the ***EmeraldUI*** *(*in blue) is the User Interface that wraps all the components functionality in a unique User Interface. It leverages the APIs provided by the components to interchange the needed commands and information and present it in a suitable manner to the final users. It offers the required functionality for the business cases to the different roles that make use of EMERALD.

2.3 Glossary

Table 2 provides a definition of the terms used in the context of EMERALD, along with examples. The definitions could be improved in the course of the project and new terms added, if needed. Therefore, these definitions will be collected in a separate document within the scope of Task 1.2 Architecture, open for all partners to contribute and consult.

Table 2. EMERALD Glossary

Term	Definition
Assurance Level	Ground for confidence that an ICT process, product, or service meets the security requirements of the European Cybersecurity Certification Scheme (EUCS) and states at what level it has been evaluated. The EU Cybersecurity Act defines the following assurance levels: • Basic

Term	Definition
	• Substantial • High *Source*: EU Cybersecurity Act [6]
Audit Scope	The audit scope refers to the scope of an individual audit. It includes the certification target (i.e., a sub-selection of resources) set into context of a particular certification scheme.
Evidence	Existence or verity of something. Can be obtained through observation, measurement, test, or by other means. Evidence for the purpose of an audit generally consists of records, statements of fact or other information which are relevant to the audit criteria and verifiable. *Examples*: • Terraform template for VM being assessed. • Audit logs from S3 bucket. • Documented security policy and procedures of a CSP. *Source*: ISO 9000 [7]
Certification Target	The target of certification comprises all entities in the cloud service that are potentially relevant for a certification. This includes (cloud) infrastructure components, the source code or binary code of deployed services, documents detailing processes as well as specific data, for example AI models.
Cloud Service	One or more capabilities offered via cloud computing invoked using a defined interface. *Source*: ISO/IEC 17788 [8]
Cloud Service Provider (CSP)	Company which makes cloud services available. *Source*: ISO/IEC 17788 [8]
Component	Any part of the EMERALD ecosystem than has a specific functionality and can be considered a separate entity with respect to other components. It is usually represented by a box in the EMERALD components diagram.
Resource	Component of the Cloud Service, which offers a specific capability to the cloud customer. *Examples:* Virtual Machines, Kubernetes clusters, Databases. *Source*: Leverages ISO 17788 [8]
Security Assessment Result	The outcome of a performed Security Assessment Rule *Example*: Compliant, Non-compliant
Security Assessment Rule	The process that applies a specific Metric to assess if the Cloud Service's configuration is compliant with a specific Target Value. The Security Assessment Rule compares a Measurement Result with the specific Target Value to obtain a Security Assessment Result. The security assessment rule is instantiated from a template which references the Metric to apply, but not the specific Target Value to use for the assessment of the Security Configuration. *Examples*: • Check the configured TLS Version of an Application Service is at least 1.2 Check the maximum password age on a cloud Linux VM is set to 30 days.
Security Evaluation Result	The result of consolidating all the Assessment Results of a given Audit Scope. The result is a yes/no, that is, the acceptation or the rejection of the certification in course according to the established rules.

Term	Definition
Security Control	A safeguard or countermeasure prescribed for an information system or an organization, designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements (cf. Technical and Organizational Measures). *Source*: Security and Privacy Controls for Federal Information Systems and Organizations - NIST Special Publication 800-53. rev 5 [9] Each Security Control has an Objective, that is, a statement describing what it is to be achieved as a result of implementing a control. *Example*: (CKM-01 POLICIES FOR THE USE OF ENCRYPTION MECHANISMS AND KEY MANAGEMENT) Objective: Policies and procedures for encryption mechanisms and key management including technical and organisational safeguards are defined, communicated, and implemented, in order to ensure the confidentiality, authenticity and integrity of the information. *Source*: ISO/IEC 27000:2018 - Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary [10] Controls exist mostly in natural language within various security frameworks and standards like the EUCS. In EMERALD, a control refers to a specific countermeasure designed to protect cloud services. We follow the definition used in OSCAL[2]: *"A control is a requirement or guideline, which when implemented will reduce an aspect of risk related to an information system and its information"*. Note that the naming of a control also differs from security standard to security standard, e.g., in the EUCS there are controls and requirements, where a control provides a more abstract description and puts multiple requirements together, while a requirement gives a concrete definition of a countermeasure. A metric, on the other hand, refers to a rule (in fact, a measurable value) used to assess one or more properties of a control. Source: NIST, Key Concepts and Terms Used in OSCAL [11]
Security Measurement Result	The outcome of measuring a Metric. *Examples*: • TLS Version = 1.0, • Maximum Password Age = 20 days, • Password Length = 6 characters, Encryption at rest = Enabled
Security Metric	A standard of measurement that describes the conditions and the rules for performing a measurement of a property and for understanding the results of a measurement. Note: The metric describes what the result of the measurement means, but not how the measurement is performed. Note: A metric is applied in practice within a given context that requires specific properties to be measured, at a given time(s) for a specific objective. *Examples*: TLS Version, Maximum Password Age, Password Length, Retention Time *Source*: NIST SP500-307 [12]
Tamper proof	Feature of the Digital Audit Trail system (DAT) guaranteeing information cannot be modified (i.e., it is impossible to be changed).
Target Value	A property of a Security Assessment Rule, defining the value for a specific Metric so the Security Configuration of the Cloud Service is compliant with the Security Requirement. The target value is defined by the CSP.

[2] https://pages.nist.gov/OSCAL/resources/concepts/terminology/#control

Term	Definition
	Example: Max Password Age <= 90 days, TLS Version In Use >= 1.2, Encryption Key Length >= 1024 bits, Retention Time > 35 days
Tool	A software element that has several disparate functions and therefore can be composed by several components. It can be seen as an aggregation of components. *Example*: Clouditor is a **tool**, and it can be composed by several **components**, like Orchestrator, Evidence Store, Evaluation, Assessment…

3 EMERALD Framework Requirements

In this chapter, we will list and jointly analyse the requirements gathered for the components during the first year of the project. During this time, several workshops have been maintained among the work packages to coordinate the different views that stakeholders could have about what the EMERALD framework must provide and how. One of the outcomes are the requirements presented here.

Each requirement is uniquely identified by an ID, which will be referenced in future tasks and documents for prioritization, validation, etc. Please note that the requirements are not described in detail, i.e., using the template defined in Table 3, because they have already been described in detail in those deliverables describing the respective EMERALD component in WP2 (see D2.2 [13], D2.4 [14], D2.6 [15], D2.8 [16]), WP3 (see D3.1 [17]), WP4 (see D4.1 [4]) and WP5 (see D5.1 [18]).

3.1 Methodology and Tools for requirements elicitation

In this section, we will briefly describe the methodology used in EMERALD for the elicitation of requirements and the principal tools and artifacts used to support the process.

3.1.1 The process

The requirements gathering process followed in EMERALD is multi-focused. The process has been divided in three parallel paths, each one trying to investigate the EMERALD system from different perspectives.

A first path that uncovers the functionalities and qualities that the technicians understand the EMERALD product has to offer. This work has been based in the documentation available: project proposal [19], key Results expected, norms and standards, and in the knowledge inherited from the MEDINA[3] project, which is the predecessor of EMERALD project. This path, carried under WP1, has produced a set of **Technical requirements.** These requirements have been covered in different deliverables in WP2 (D2.2 [13], D2.4 [14], D2.6 [15], D2.8 [16]) and WP3 (D3.1 [17]) devoted to describing the components.

A second path has been devoted specifically to the user experience, to provide EMERALD with an advanced user interface that connects the rest of components and satisfies the users' requirements while providing the needed information in its different views. This work has been conducted in WP4, where a co-design, participatory design approach has been followed, holding separate interviews with component owners and with pilot owners. This has produced a set of **User Interface requirements** (more information on this is available in D4.1 [4]).

Lastly, a third path has been focused on what the final users of EMERALD have asked to be part of the delivered product. This work has been part of WP5, where the pilots have been defined, and has produced a set of **Business requirements** (more information on this is available in the deliverable D5.1 [18]).

All these separate elicitations have produced separate requirement sets. One of the tasks in WP1 has been to analyse, refine and check these requirements, approve the correct ones and discard others, as well as to establish the relationships among them. Several discussions about the requirements have hold during the periodic work package meetings. Also, specific workshops have been conducted to map the business requirements and user interface

[3] https://medina-project.eu/

requirements to technical requirements. This has produced changes in the requirements, and new requirements have been defined when necessary.

This document presents the results of this analysis, managing the different lists of requirements. We provide a dashboard with the status and prioritization of requirements. Furthermore, several traceability matrixes are presented, to keep all the relationships affecting the requirements up to date.

3.1.2 The tools

To carry out the architecture definition, different tools and artifacts have been used, namely Gitlab issues, Component cards and UML models with PlantUML tool, that will be briefly described in the following.

3.1.2.1 Gitlab issues

To better control their changes and evolution, the requirements in EMERALD have been defined in GitLab, using the issues[4] feature. Issues are used in general to collaborate on ideas, solve problems, and plan work. They allow to track tasks and work status, accept feature proposals, ask questions, or support requests.

A template has been used to define the requirements, as depicted in Table 3. The template has a tabular form and contains all the fields needed to gather the requirement information and track it during the project lifetime. The table has been also implemented as a GitLab template, useful to define new requirements.

Table 3. Requirement template

Field	**Description**
Requirement ID	Unique identifier. E.g., for the Repository of Controls and Metrics -> RCM.01, RCM.02…
Short title	Short description of the requirement
Description	More detailed description of the requirement. This is especially relevant for the creation of the test cases.
Status	Choose the corresponding label: Status::Proposed -> Status::Accepted / Status::Discarded -> Status::Work in Progress -> Status::Implemented -> Status::Validated
Priority	Choose the corresponding label: Priority::Must -> Priority::Should -> Priority::Could
Component	Choose the corresponding label: Comp::AI-SEC, Comp::AMOE, Comp::CertGraph, Comp::Clouditor, Comp::Codyze, Comp::eKnows, Comp::EmeraldUI, Comp::EvidenceStore, Comp::LCM, Comp::RCM, Comp::RMA, Comp::TWS, Comp::WP1, Comp::N/A
Source	Pilots / Component / DoA / KPI
Type	Choose the corresponding label: Choose the corresponding label: Type::Technical, Type::Pilots, Type::GUI

[4] https://docs.gitlab.com/ee/user/project/issues/

Related KR	Choose the corresponding label: KR::KR1_EXTRACT, …, KR::N/A
Related KPI	Choose the corresponding label: KPI::1.1, …, KPI::N/A
Validation acceptance criteria	Describe how to validate the requirement. What are the steps to follow, what should be the system output
Progress	[Optional] percentual degree of advances from 0% to 100%
Milestone	Select the milestone among the defined ones: from MS1: Components V1 (M12) to MS9: Final evaluation report and impact analysis (M36)

As mentioned above, this table has been used in other WP2 and WP3 deliverables dedicated to describing the components in detail. In this document, we will mainly limit to listing the requirements and analysing them as a whole.

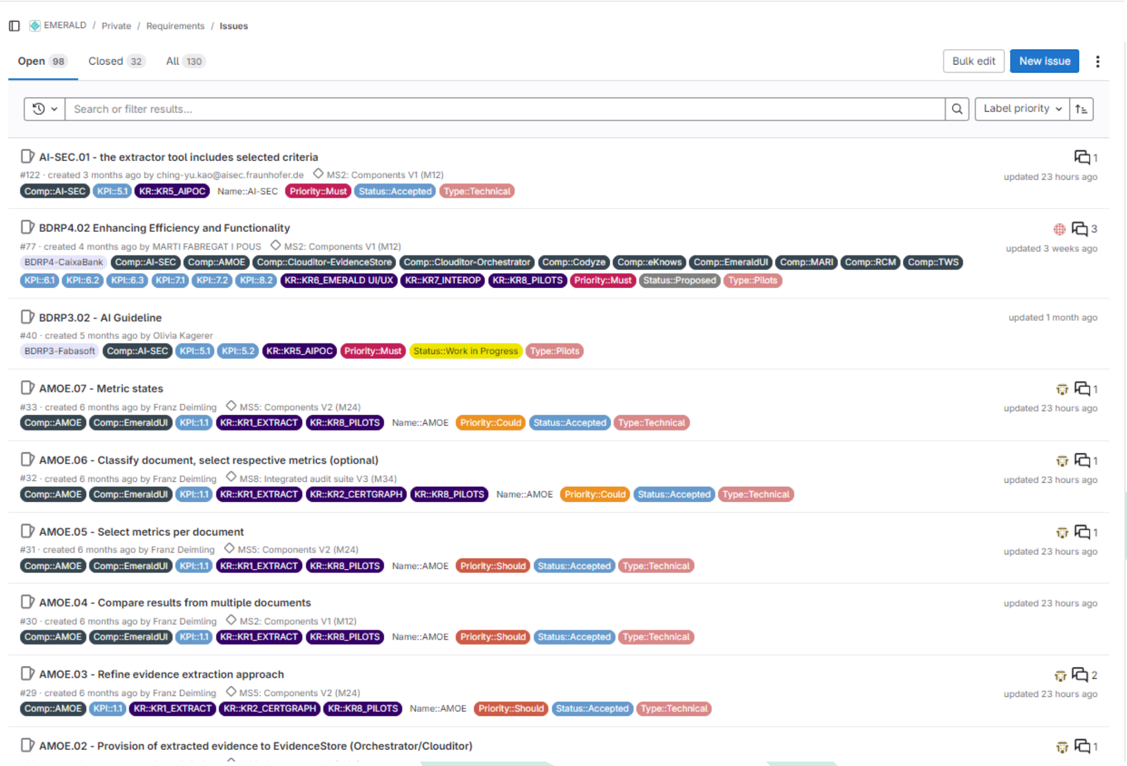
Figure 3 shows a list of the requirements in the GitLab requirements repository. The developer can define a new requirement using the aforementioned template. To facilitate requirements identification and filtering, a set of labels associated to the issues have been defined. Labels are organized in categories, where each category defines a property of the requirement and is represented in different colours. Categories for labels are:

- Component label (one for each component)
- Type label (Technical / Pilots, UX)
- Priority label (Must / Should / Could)
- KR label (one for each Key Result)
- Pilot label (Ionos / CloudFerro / Fabasoft / Caixabank)
- Status label (Proposed / Accepted / Discarded / Implemented / Validated)
- KPI label (one for each Key Performance Indicator)

Requirements can be filtered using lists or also be visualized and managed using issue boards[5] of GitLab. The issue board is a software management tool used to plan, organize, and visualize a workflow for a feature or product release, pairing issue tracking and project management. The boards organize the issues in cards, in vertical lists organized by their labels, milestones, or assignees. Requirements can be managed inside the boards. For example, moving a requirement from one list to other changes the associated label and thus the requirement properties. Several specific boards have been defined in EMERALD to provide different views of the requirement set:

- Requirements by TYPE(Technical/GUI/Pilots)
- Requirements by PRIORITY(Must/Should/Could)
- Requirements by KR
- Requirements by STATUS
- Requirements by COMPONENT
- Requirements by Pilot

[5] https://docs.gitlab.com/ee/user/project/issue_board.html



EMERALD / Private / Requirements / Issues

Open 98 Closed 32 All 130

Bulk edit New issue

Search or filter results... Label priority

AI-SEC.01 - the extractor tool includes selected criteria
#122 · created 3 months ago by ching-yu.kao@aisec.fraunhofer.de ◇ MS2: Components V1 (M12)
Comp::AI-SEC KPI::5.1 KR::KR5_AIPOC Name::AI-SEC Priority::Must Status::Accepted Type::Technical
1 updated 23 hours ago

BDRP4.02 Enhancing Efficiency and Functionality
#77 · created 4 months ago by MARTI FABREGAT I POUS ◇ MS2: Components V1 (M12)
BDRP4-CaixaBank Comp::AI-SEC Comp::AMOE Comp::Clouditor-EvidenceStore Comp::Clouditor-Orchestrator Comp::Codyze Comp::eKnows Comp::EmeraldUI Comp::MARI Comp::RCM Comp::TWS
KPI::6.1 KPI::6.2 KPI::6.3 KPI::7.1 KPI::7.2 KPI::8.2 KR::KR6_EMERALD UI/UX KR::KR7_INTEROP KR::KR8_PILOTS Priority::Must Status::Proposed Type::Pilots
3 updated 3 weeks ago

BDRP3.02 - AI Guideline
#40 · created 5 months ago by Olivia Kagerer
BDRP3-Fabasoft Comp::AI-SEC KPI::5.1 KPI::5.2 KR::KR5_AIPOC Priority::Must Status::Work in Progress Type::Pilots
updated 1 month ago

AMOE.07 - Metric states
#33 · created 6 months ago by Franz Deimling ◇ MS5: Components V2 (M24)
Comp::AMOE Comp::EmeraldUI KPI::1.1 KR::KR1_EXTRACT KR::KR8_PILOTS Name::AMOE Priority::Could Status::Accepted Type::Technical
1 updated 23 hours ago

AMOE.06 - Classify document, select respective metrics (optional)
#32 · created 6 months ago by Franz Deimling ◇ MS8: Integrated audit suite V3 (M34)
Comp::AMOE Comp::EmeraldUI KPI::1.1 KR::KR1_EXTRACT KR::KR2_CERTGRAPH KR::KR8_PILOTS Name::AMOE Priority::Could Status::Accepted Type::Technical
1 updated 23 hours ago

AMOE.05 - Select metrics per document
#31 · created 6 months ago by Franz Deimling ◇ MS5: Components V2 (M24)
Comp::AMOE Comp::EmeraldUI KPI::1.1 KR::KR1_EXTRACT KR::KR8_PILOTS Name::AMOE Priority::Should Status::Accepted Type::Technical
1 updated 23 hours ago

AMOE.04 - Compare results from multiple documents
#30 · created 6 months ago by Franz Deimling ◇ MS2: Components V1 (M12)
Comp::AMOE Comp::EmeraldUI KPI::1.1 KR::KR1_EXTRACT KR::KR8_PILOTS Name::AMOE Priority::Should Status::Accepted Type::Technical
updated 23 hours ago

AMOE.03 - Refine evidence extraction approach
#29 · created 6 months ago by Franz Deimling ◇ MS5: Components V2 (M24)
Comp::AMOE KPI::1.1 KR::KR1_EXTRACT KR::KR2_CERTGRAPH KR::KR8_PILOTS Name::AMOE Priority::Should Status::Accepted Type::Technical
2 updated 23 hours ago

AMOE.02 - Provision of extracted evidence to EvidenceStore (Orchestrator/Clouditor)
1

Figure 3. List of requirements as issues in GitLab (excerpt)

3.1.2.2 Component cards

A “component card” is what we call a piece of information that contains a brief description of each component. It contains the essential information to know **what** the component does, **where** it fits in the framework, **with which** other components it interacts and **how** it is made.

A component card has been defined for each component, and all of them are included as part of the detailed view of the EMERALD framework in Section 4. Table 4 shows the structure of a component card.

Table 4. Component card template

<table>
<tr><td><b>Component Name</b></td><td>Name of the component and acronym, if any</td></tr>
<tr><td><b>Main functionalities</b></td><td>List the main functionalities the component provides. E.g.:
• Describe functionality 1
• Describe functionality 2</td></tr>
<tr><td><b>Sub-components Description</b></td><td><b>Subcomponent A:</b> Describe the functionality of the sub-component
<b>Subcomponent B:</b></td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td>Include graphical interfaces if any.
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

Interaction with other components	• **Component X**: Describe interaction with component X • **Component Y**:
Relevant sequence diagram/s	Include a shot of the sequence diagram(s) describing the component's dynamic behaviour
Requirements Mapping	List the requirements covered by this component. E.g.: • TWS.01: Provide integrity proof of evidence • TWS.02:
Technology used	Describe the technology used in the implementation of the component (languages, frameworks, etc)
Related KR	Related EMERALD proposal Key Results
WP and task	WPX – Tx.1
License	License of the component
Partner	Partner that is the component owner, who defines/implements it.

3.1.2.3 PlantUML diagrams

Diagrams of the Unified Modelling Language (UML) have been used in the definition of the EMERALD architecture. More concretely, **Class diagrams** to define the data model the components use, and the relationship among the objects; and **Sequence diagrams**, to define the dynamic behaviour of the components and the flow of information among them. This kind of diagram visualizes the interactions between users, systems and sub-systems over time, through message passing between objects or roles. UML sequence diagram complete the classes or object diagram, that represent the attributes, by representing the programming logic to be filled in the methods' body.

To define the UML diagrams, the PlantUML[6] tool was chosen. This tool creates the diagram based in text descriptions and supports a wide range of diagrams. PlantUML allows to render the diagrams as images in different output formats. As the PlantUML based diagrams contain text/code, the files are included in Gitlab for versioning. This allows for different organisational processes, that are not possible in common online tools with graphical support. New versions of the diagrams are produced with each commit, and merge requests are created to change the actual release.

As the specific diagram for each component has been included in the deliverable D1.1 [1], in this document we only present a general class diagram representing the whole EMERALD framework. However, sequence diagrams for each component are included in Section 4 as part of the detailed view of the EMERALD framework.

3.2 Functional Requirements

Table 5 lists the set of functional requirements of the EMERALD framework components. Along with the brief description, the priority and milestone of each requirement are presented. A total of 44 functional requirements have been elicited, grouped in the 12 components that form the framework.

[6] https://plantuml.com/

The Identification of each requirement is unique. It is composed by the acronym of the component plus a number. The components have been described in Section 2.2, but a list with its correspondence to Identifiers is provided below for clarity.

- AI-SEC: *AI Security Evidence Collector*
- AMOE: *Assessment and Management of Organisational Evidence*
- CLDISC: *Clouditor-Discovery*
- CODYZE: *Codyze*
- EKNOWS: *eknows* - Software analysis platform
- TWS: *Trustworthiness System*
- MARI: *Mapping Assistant for Regulations with Intelligence*
- RCM: *Repository of Controls and Metrics*
- ORCH: *Clouditor-Orchestrator*
- ESTORE: *Clouditor-Evidence Store*
- ASSESS: *Clouditor-Assessment*
- EVAL: *Clouditor-Evaluation*

The Milestone field of each requirements signals when the requirement is foreseen to be completed. The list of Milestones corresponds to the ones defined in the DoA:

- MS1: Project baselines and definition (M9)
- MS2: Components V1 (M12)
- MS3: Integrated audit suite V1 (M18)
- MS4: Pilots V1 (M20)
- MS5: Components V2 (M24)
- MS6: Integrated audit suite V2 (M30)
- MS7: Pilots V2 (M32)
- MS8: Integrated audit suite V3 (M34)
- MS9: Final evaluation report and impact analysis (M36)

Table 5. Functional requirements.

Req. ID	Description	Priority	Milestone
AI-SEC.01	**The extractor tool includes defined criteria:** The designed AI-SEC has the selected criteria of the BSI AIC4	Must	MS2 (M12)
AMOE.01	**Upload PDF document:** The component shall be able to receive a PDF document via API and process its contents regarding the defined metrics. The PDF shall receive a unique ID so that it can be retrieved and deleted later on.	Must	MS2 (M12)
AMOE.02	**Provision of extracted evidence to EvidenceStore:** The evidence extraction component needs to be able to forward the extracted evidence to the EMERALD EvidenceStore, so it can be used for assessment and further audit processes.	Must	MS5 (M24)
AMOE.03	**Refine evidence extraction approach:** The evidence extraction approach should be refined to the needs of the pilots, so that the tool is able to provide relevant evidence for the metric assessments.	Must	MS5 (M24)
AMOE.04	**Compare results from multiple documents:** Results from multiple policy documents shall be comparable using AMOE. A metric can be used to extract evidence from different policy documents. AMOE shall provide the results via API for a metric and given cloud service.	Should	MS2 (M12)

Req. ID	Description	Priority	Milestone
AMOE.05	**Select metrics per document:** AMOE should offer the possibility to select some metrics before they are extracted for a document. This speeds up the processing time as metrics that are not contained in the document do not need to be checked. Also, it should be more convenient for the user, as the results are more precise and less irrelevant results need to be discarded.	Should	MS5 (M24)
AMOE.06	**Classify document, select respective metrics (optional):** AMOE could use document classification to pre-select some metrics based on the category, text, requirements or other feature that would be of use. This could potentially, reduce the manual workload and help to provide only results for metrics that target the specific document.	Must	MS8 (M34)
AMOE.07	**Metric states:** AMOE could add some internal states to the metrics. This should help to visualize the current process for every metric and role. Here is a list of metric flags that could be used: new, internal-started, ready-for-audit, revise-policy, audit-finished, result-outdated, extraction-failed. - new: the metric has been successfully extracted - extraction-failed: evidence could not be extracted - internal-started: internal auditor/compliance manager started inspecting the metric - ready-for-audit: internal auditor/compliance manager has finished with the metric, and marked it ready for auditor - revise-policy: auditor sets metric to be revised - audit-finished: auditor is ok with metric - result-outdated: automatic or manual triggered check if result is outdated	Should	MS5 (M24)
CLDISC.01	**Discovery of security properties of infrastructure components:** The Clouditor discovery needs to discover security properties of infrastructure components. The evidence with the security properties is sent to the Evidence Store in the ontology format.	Must	MS6 (M30)
CODYZE.01	**Extraction of security features from source code:** Codyze needs to check available source code artefacts for security features.	Must	MS6 (M30)
EKNOWS.01	**Integration into existing systems:** The component should be integrable into existing systems, development environments and workflows, for example by using APIs like REST by compatibility with CI/CD-Pipelines.	Must	MS3 (M18)
EKNOWS.02	**Resilience while analysing erroneous code:** The source code analysed by the component could be erroneous, for example syntactical and semantical errors could be encountered while parsing it. Furthermore, an unknown dialect of a language could be encountered. An appropriate error handling strategy for such situations is necessary: Erroneous code will be skipped and not be further analysed. A corresponding error message will be stored in the gathered evidence.	Should	MS5 (M24)
EKNOWS.03	**Multi-language support:** The component should be able to analyse source code written in different programming languages and should support at least Java and Python.	Must	MS5 (M24)
EKNOWS.04	**Support EMERALD evidence format:** The analyzation results are offered in a structured and standardized format, the EMERALD evidence format (see data model). This enables further processing and queries in other components.	Must	MS3 (M18)

Req. ID	Description	Priority	Milestone
EKNOWS.05	**Static code analysis:** The component uses static code analysis methods. Such methods are, for example, data flow analysis, call graph analysis, symbolic execution or control flow analysis. One or multiple methods (possibly in combination) will be used to gather evidence. The actual used method(s) depend(s) on the metric, for which evidence should be extracted.	Must	MS5 (M24)
TWS.01	**Provide a tool allowing the verification of evidence integrity** without needing to store the evidence itself (for confidentiality reasons).	Must	MS2 (M12)
TWS.02	**Provide a tool allowing the verification of assessment results** integrity without needing to store the result itself (for confidentiality reasons).	Must	MS2 (M12)
TWS.03	The **integrity validation** of evidence and assessment results must be done through REST API or graphical interface (EMERALD UI).	Must	MS5 (M24)
TWS.04	The TWS must be **based on a real Blockchain network**, with multiple nodes and multiple organizations to guarantee suitable decentralization and governance of the Blockchain network.	Must	MS5 (M24)
MARI.01	**AI-based:** MARI is a tool based on state-of-the-art artificial intelligence, e.g., uses a transformer-based architecture	Must	MS6 (M30)
MARI.02	**Automatic association:** MARI takes as input cloud security controls written in natural language, metrics that validate those controls, again written in natural language, and automatically returns as output the association control/metric(s) and the association control/control.	Must	MS6 (M30)
MARI.03	**Performance Evaluation:** The performance of MARI should improve on the performance of the Metric Recommender of EMERALD's predecessor project, MEDINA. We can assume that we measure the performance of MARI with the same metrics used for the Metric Recommender, namely precision@k and NDCG (Normalised Discounted Cumulative Gain)	Must	MS6 (M30)
MARI.04	**Usage and Visualization:** MARI should be invoked through EMERALD's built-in interface, and MARI results can be visualized through the same interface	Must	MS6 (M30)
MARI.05	**Strategies**: MARI can act according to specific strategies, such as considering only technical controls, or organizational controls, or controls of a certain category, or controls whose implementation costs less in terms of human resources, etc. The strategies will be defined during the project.	Must	MS6 (M30)
RCM.01	**Multi-schema support**: The repository should contain at least an additional security scheme, apart from the EUCS that is the scheme implemented in MEDINA Catalogue and is inherited in EMERALD	Must	MS2 (M12)
RCM.02	**Accessible by the rest of components**: The repository content should be made accessible to the rest of EMERALD components via API	Must	MS2 (M12)
RCM.03	**Include metrics for all schemes supported**: The repository should include metrics that could be used to assess the compliance with one or more certification schemes	Must	MS2 (M12)
RCM.04	**Mapping of schemes**: The repository should support the mapping of the certification schemes contained. The scheme-to-scheme mapping will be provided by the MARI tool and stored in the repository. The rationale for the mapping decision will also be stored	Should	MS5 (M24)

Req. ID	Description	Priority	Milestone
RCM.05	**Import/export of security schemes in OSCAL**: The repository is able to import a new scheme defined in the OSCAL language (this feature can also be used to update an existing scheme). The repository is able to export any available scheme in OSCAL format	Must	MS6 (M30)
RCM.06	**Import/export of security schemes in CSV format**: The repository can export a scheme to a CSV file, and import a CSV file with the same format as a new scheme	Could	MS2 (M12)
RCM.07	**Support for personalized catalogues**: The Repository has to offer the user the possibility to create a personalized catalogue of controls. These controls can be taken from the same or from different security schemes	Must	MS6 (M30)
RCM.08	**Support updating/versioning of schemes**: The Repository has to maintain a versioning system of the schemes it contains, so that if a new version is uploaded, it is able to detect the change and notify the user that a new version is available	Should	MS6 (M30)
ORCH.01	**Final certificate decision:** Since we do not have a dedicated life-cycle manager component in EMERALD, the Orchestrator must take care of the final certificate decision. The decision is based on the input of the Evaluation component providing the Orchestrator with an evaluation result for each control	Must	MS5 (M24)
ORCH.02	**REST API Gateway for UI:** The Orchestrator should provide a REST API gateway for the UI that serves a central API endpoint for all information needed from the Orchestrator, Assessment, Evaluation and other Clouditor components.	Must	MS2 (M12)
ORCH.03	**Role Based Access Control (RBAC):** Since the UI wants to selectively disclose information to users and/or roles, we need a RBAC mechanism in our API endpoints, mainly in the Orchestrator.	Must	MS5 (M24)
ORCH.04	**Manage Tools via API**: We need to manage external tools, such as evidence extractors in the Orchestrator.	Should	MS5 (M24)
ORCH.05	**Provide an API for audit workflow:** We want to assign people to controls within an audit instance that have a particular task.	Must	MS6 (M30)
ESTORE.01	**Storage of evidence as ontology entities in graph database:** The Evidence Store must store the evidence according to the schema defined by the knowledge graph. The preferred way to store this information is a graph database.	Must	MS3 (M18)
ESTORE.02	**Allow Interaction with Third-Party Tools:** The Evidence store should be allowed to accept evidence from third-party tools, e.g., using a REST API. The evidence needs to be in the ontology format. Therefore, information about the ontology and data models must be available.	Should	MS3 (M34)
ASSESS.01	**Assessment based on evidence:** The assessment should assess evidence based on the knowledge graph.	Must	MS6 (M30)
ASSESS.02	**Assessment rules for 80% of the defined metrics:** Assessment rules must exist for 80% of the metrics defined in KPI4.1.	Must	MS6 (M30)
ASSESS.03	**Display cause of assessment result:** We want to know why an assessment result fails or passes.	Could	MS6 (M30)
EVAL.01	**Display cause of failing evaluation result:** We want to know why the evaluation result fails or passes. Therefore, it should contain a list of assessment results that cause the evaluation status to be non-compliant.	Could	MS6 (M30)
EVAL.02	**Evaluation based on assessment results:** The evaluation should assess the result based on all the required assessment results stored in the database.	Must	MS6 (M30)

3.3 Non-Functional Requirements

The technical requirements presented in Section 3.2 involve behavioural, or functional, requirements of the system. They tell us how the system must behave when presented with certain inputs or conditions.

But, in addition to these functional requirements, we have defined some non-functional requirements for the EMERALD framework. The following subsections provide different types of non-functional requirements, gathered in different work packages.

3.3.1 Other WP1 requirements

We present here a list of nonfunctional requirements defined in WP1. These requirements are related with characteristics or constrains of the system more that to its behaviour. They have not been included in any previous deliverables, so we follow each requirement with a short paragraph on how we plan to implement it.

Requirement id	**WP1.01**
Short title	Performant framework
Description	The EMERALD framework should be as performant as possible. The response time for a user action in normal conditions should not be larger than a few seconds.
Implementation state	Partially implemented

The component tools will have to pass automatic integration tests by the CI/CD pipeline before being integrated into the framework. The validation task in WP5 will validate both the functionality and the performance of the EMERALD framework. Apart of these controls, the framework infrastructure is continuously monitored, and the implemented environment allows flexibility to upgrade the resources if they are falling short (e.g., adding more memory or CPUs to the Kubernetes nodes, or providing extra nodes).

Requirement id	**WP1.02**
Short title	Portability
Description	The EMERALD framework should be portable and work in any typical business environment.
Implementation state	Partially implemented

The components of the framework will be packaged as containers, which are a portable technology by definition. We will use the *Docker* ecosystem to build and share images. For image building we will support both *Docker* and *Docker Compose.*

Requirement id	**WP1.03**
Short title	Scalability
Description	The EMERALD framework should be easily scalable when the working conditions become severe in relation to the number of users of the platform or intense use.
Status	Partially implemented

Scalability will be based in the use of a container orchestration technology, such as Kubernetes, which is inherently scalable. It also can provide resilience, helping to solve problems when the resources allocation is shorter that needed.

Requirement id	**WP1.04**
Short title	Installability

Description	The EMERALD framework has to be easy to install. There must exist documents that facilitate the installation procedure.
Implementation state	Partially implemented

The EMERALD environment will be defined using Infrastructure as code (IaC). By now, the integration environment is defined, composed by a four-node *Kubernetes* cluster -configured by a set of *Ansible* playbooks- over *vSphere* platform.

Requirement id	**WP1.05**
Short title	Documentation
Description	All the components of the EMERALD framework will provide associated documentation, covering as a minimum the installation, how to use and the license.
Implementation state	Partially implemented

During the project, software type deliverables will always include a companion document to specify the characteristics of the software. Part of this document will the user manual or the instructions for use the software.

Requirement id	**WP1.06**
Short title	Agile development
Description	The EMERALD framework will be constructed using an agile methodology, with several cycles of Design, Build, Test, and Deploy.
Implementation state	Fully implemented

The management of the project has already foreseen three incremental releases -V1, V2, V3- in months M12, M24 and M33. The WP1 team will provide several tools to make this possible, for example:

- **Source control**: *GitLab* tool allows code management and implementation of CICD processes that help to speed up the development.
- **CI/CD processes**: *GitLab CI* allows for continuous integration and deployment tasks to be implemented.
- **Integration automation**. A *GitLab Agent* for *Kubernetes* monitors the framework repository and will allow deploying and testing new versions of the components directly, checking the health of the components.

Requirement id	**WP1.07**
Short title	Observability[7]
Description	Monitoring mechanism have to be provided to measure the health of the EMERALD Framework.
Implementation state	Partially implemented

Monitoring will be provided based in the Kubernetes dashboard and the log system features. This will provide almost instantaneous feedback on the system health, and also access to logs of the different components in order to recognize the status of the system and detect possible problems.

Requirement id	**WP1.08**

[7] A system is said to be observable if, for every possible evolution of state and control vectors, the current state can be estimated using only the information from outputs. In other words, one can determine the behaviour of the entire system from the system's outputs (wikipedia)

Short title	Security
Description	The EMERALD framework has to be secure. This implies correct user authentication and authorization, secret management, preventing intrusion, etc.
Implementation state	Partially implemented

For the user management, a specific tool as Keycloak[8] will be installed, which is specifically designed to manage identity and access. Keycloak supports OpenID Connect, single-sign-on for all the components and allows the synchronization with external identity sources. The framework will implement role-based access control as authorization mechanism to avoid every user has access to every functionality. The system will store API keys, certificates, and passwords as Kubernetes Secrets, which it will then add to the pods. In WP1, we will implement network policies - using the Traefik inverse proxy - to limit how containers and services talk to each other inside a Kubernetes cluster, which reduces the ways attackers could get in.

3.3.2 Business driven requirements

The business-driven requirements have been worked and defined by the individual pilots in Task 5.1 of WP5 and are available in the deliverable D5.1 [18] for each of the pilots. Table 6 provides a summary list of these requirements for completion and reference, before to proceed with the analysis of requirements in Section 3.4.

Table 6. Business driven requirements

Req. ID	Description	Priority
BDRP1.01	Automate and Streamline Certification Processes	Must
BDRP1.02	Secure and Reliable Long-term Evidence Storage	Must
BDRP1.03	Efficient Requirement and Compliance Mapping	Must
BDRP1.04	Central Management of Controls and Metrics	Must
BDRP1.05	Compliance Verification for Organizational Policies	Must
BDRP1.06	Ensure Software Compliance through Static Code Analysis	Must
BDRP1.07	Intuitive User Experience for Compliance Monitoring	Must
BDRP2.01	OpenStack	Must
BDRP2.02	Reusable Metrics & Requirements	Must
BDRP2.03	Transparency increase	Must
BRDP2.04	Intuitive UI	Must
BDRP2.05	Security Schemes for Pilot 2	Must
BDRP3.01	UI/UX Concept	Must
BDRP3.02	AI Guideline	Must
BDRP3.03	Integration of Internal evidence collection tools	Must
BDRP3.04	Reusable Metrics	Must
BDRP3.05	Security Schemes Pilot 3	Must

[8] https://www.keycloak.org/

Req. ID	Description	Priority
BDRP3.06	Custom set of requirements	Must
BDRP3.07	Enhance current audit process	Should
BDRP3.08	Audit Transparency	Should
BDRP3.09	Manual controls	Should
BDRP3.10	Safe security scheme updates	Should
BDRP3.11	Checks for policy documents	Must
BDRP3.12	Use of standard for export/import	Should
BDRP4.01	Broad Usability & BYOCS (Bring You Own Certification Scheme)	Must
BDRP4.02	Enhancing Efficiency and Functionality	Must
BDRP4.03	Ensuring Traceability for Certificates and Audits	Must
BDRP4.04	User-Friendly Interface for All Employees	Should
BDRP4.05	Integration with Internal Tools	Must
BDRP4.06	Seamless Migration and Integration	Must
BDRP4.07	Documentation	Should

3.3.3 UI/UX requirements (usability)

The User Interface/User Experience requirements have been developed and defined in the WP4, and the complete description is available in the deliverable D4.1 [4]. We have extracted the list that is shown in Table 7 for completion and reference.

Table 7. UI/UX requirements

Req. ID	Description	Priority
UIUX.01	Landing Page	Must
UIUX.02	Audit Instance Creation View	Must
UIUX.03	Requirements Overview View	Must
UIUX.04	Requirements Overview View: Progress Indicators	Must
UIUX.05	Requirements Overview View: Filtering and Searching	Must
UIUX.06	Policy Documents Manager View	Must
UIUX.07	Policy Documents Manager View: Metrics Selection	Should
UIUX.08	Evidence Extractors View	Must
UIUX.09	Requirement Detail View	Must
UIUX.10	Requirement Detail View: Assignment	Must
UIUX.11	Requirement Detail View: History	Must
UIUX.12	Requirement Detail View: Evidence	Must
UIUX.13	Requirement Detail View: Non-Compliance	Must

Req. ID	Description	Priority
UIUX.14	MARI Tool View	Must
UIUX.15	Certification Schemes Manager View	Must
UIUX.16	Certification Schemes Manager View: BYOCS	Must
UIUX.17	Certification Schemes Manager View: Import/Export	Could
UIUX.18	Trustworthiness Check	Must
UIUX.19	Intuitive and Smooth UI	Must
UIUX.20	Reusable metrics	Must
UIUX.21	Transfer of Audit to EMERALD	Should
UIUX.22	Requirement Detail View: Manual Evidence	Should
UIUX.23	Import/Export of information	Should

3.4 Analysis of Requirements

In this section we will examine the functional requirement list from different perspectives, to gain some insight about how the requirements represent the solution that EMERALD tries to build.

3.4.1 Mapping of requirements to KRs

The functional requirements have been defined in Section 3.2. The map among requirements and Key Results (KRs) offers a view on how the KRs are covered by the requirements.

Let's first present the Key Results of the EMERALD project, as were defined in the DoA [19]. The description of the Key Results is included below the mapping.

- **KR1**: EXTRACT
- **KR2**: CERTGRAPH
- **KR3**: OPTIMA
- **KR4**: MULTICERT
- **KR5**: AIPOC
- **KR6**: EMERALD UI/UX
- **KR7**: INTEROP
- **KR8**: PILOTS

The mapping is shown in Table 8. As a first sight, it can be affirmed that all the elicited Functional Requirements are related to one or more KRs (note that "KR8: Pilots" is not included in the table, as the relation with the pilot is addressed in more detail thereafter).

Table 8. Functional requirements and KRs alignment matrix

	Req. ID	KR1	KR2	KR3	KR4	KR5	KR6	KR7
1	AI-SEC.01					X		
2	AMOE.01	X	X					
3	AMOE.02	X	X					
4	AMOE.03	X						
5	AMOE.04	X	X					

	Req. ID	KR1	KR2	KR3	KR4	KR5	KR6	KR7
6	AMOE.05	X						
7	AMOE.06	X	X					
8	AMOE.07	X						
9	CLDISC.01	X						
10	CODYZE.01	X						
11	EKNOWS.01	X						
12	EKNOWS.02	X						
13	EKNOWS.03	X						
14	EKNOWS.04	X						
15	EKNOWS.05	X						
16	TWS.01							X
17	TWS.02							X
18	TWS.03							X
19	TWS.04							X
20	MARI 1.0			X				
21	MARI 2.0			X				
22	MARI 3.0			X				
23	MARI 4.0			X				
24	MARI 5.0			X				
25	RCM.01							X
26	RCM.02							X
27	RCM.03							X
28	RCM04							X
29	RCM.05							X
30	RCM.06							X
31	RCM.07							X
32	RCM.08							X
33	ORCH.01				X			
34	ORCH.02						X	
35	ORCH.03						X	
36	ORCH.04				X			
37	ORCH.05				X			
38	ESTORE.01		X					
39	ESTORE.02	X						
40	ASSESS.01				X			
41	ASSESS.02				X			
42	ASSESS.03						X	
43	EVAL.01						X	
44	EVAL.02				X			

KR1: EXTRACT: A framework to continuously extract knowledge on various layers of the cloud service (infrastructure, code, business processes) and prepare suitable evidence based on them. This result covers the improvements on existing evidence extraction tools and concepts of

MEDINA, such as AMOE. The tools enable different levels of abstraction – from low level such as source code to higher levels, such as policies and procedures.

KR2: CERTGRAPH: A graph-based structure, the certification graph, to consolidate all necessary information of the service and make it easily query-able. The graph-based approach allows storing and linking heterogeneous information extracted from different evidence sources. Furthermore, linking allows to create additional nodes in the graph that aggregate individual aspects and fragments of information to a higher-level of combined evidence, while maintaining traceability back to information sources.

KR3: OPTIMA: An intelligent system to select an optimized set of metrics that can be measured to demonstrate compliance to the selected certification scheme. One of such optimizations could be the maximum amount of re-used evidence.

KR4: MULTICERT: A tool to assess chosen metrics based on information stored in the certification graph and to evaluate the final certificate decision.

KR5: AIPOC: By transferring the innovation results to upcoming AI certification schemes, EMERALD establishes a proof of concept (PoC) on how to scale the CaaS approach to cloud-based AI systems.

KR6: EMERALD UI/UX: A user interaction concept and conducted studies to show what information each user needs in an audit process. The concept shall lead to a user interface (UI), which is tailored to the users' needs during all stages of an audit and guides them through the process of identifying problems top down – from high level requirements down to specific implementation in documents (e.g., policies) or technical specifications.

KR7: INTEROP: EMERALD will provide an interoperability layer among the trustworthy systems, assessment results and catalogue data. Security schemes are prone to change and thus updates would be required. EMERALD aims to mitigate this by incorporating the scheme data in a standardized format such as OSCAL. To enable fast development and integration of external resources, a common data format can help. Furthermore, EMERALD aims at providing interoperability at the trustworthy evidence layer by evaluating usage of the European Blockchain Services Infrastructure (EBSI) for its trustworthiness system.

KR8: PILOTS: Involvement of realistic use cases by potential applicants of EMERALD. This is key to derive and validate the proposed contents of O1 – O4. PILOTS is responsible for providing these real-world application examples and test data. The data will be forwarded to the evidence extraction stakeholders, so the components can be fine-tuned to improve quality of the results.

3.4.2 Mapping of requirements to KPIs

Similar to the KRs, the mapping of requirements and Key Performance indicators (KPIs) offers a view on how the KPIs are covered by the requirements.

This is the list of KPIs that have been defined in the DoA [19]:

- KPI 1.1: Provide support for evidence extraction from different sources (infrastructure, code, processes)
- KPI 1.2: Provide novel methods for the security assessment of AI models and their evidence generation
- KPI 2.1: Provide a schema for storing and linking heterogeneous evidence information
- KPI 2.2: Provide support traceability to information sources and extraction processes

- KPI 2.3: Provide scalability for storing/processing continuously collected evidence; demonstrated in the pilots
- KPI 3.1: Provide scheme to scheme mapping functionality based on metrics, recommended to the user
- KPI 3.2: Provide metric-to-requirement-mapping functionality by improving MEDINA approaches and incorporating KPI 5.1 results
- KPI 3.3: Provide insights for the mapping decision and how the recommendation process works
- KPI4.1: Provide realizable metrics that demonstrate compliance to at least two security certification schemes
- KPI 4.2: Provide metric assessment for 80 % of the metrics in KPI 4.1 based on the certification graph
- KPI 5.1: Provide realizable metrics to help evaluate at least 50% of the categories of criteria of the BSI AIC4 that deal with the robustness of ML system, their interpretability, and the mitigation of potentially negative impacts such as model unfairness (c.f. Chapter 6, AIC4).
- KPI 5.2: Provide a PoC for semi-automated assessment of 80% of the metrics specified in KPI 5.1.
- KPI 6.1: Provide roles and workflows, derived from interviews with relevant users (e.g., project partners and advisory board members), develop mock-ups and interaction concepts for managing the audit process
- KPI 6.2: Provide concept for the (UI) of EMERALD and integration of evidence collection components, data bases and orchestrating components
- KPI 6.3: Provide a graphical user interface for role-based access to certification information content
- KPI 7.1: Conventionalize import and export functionalities to take or share data with external sources
- KPI 7.2: Incorporate input from standardisation bodies and synchronize data formats and protocols
- KPI 8.1: Facilitate at least two different audit scenarios, one for public clouds, one for private cloud installations
- KPI 8.2: Validate user acceptance in terms of complexity reduction

Table 9. Functional requirements and KPIs alignment matrix.

Req. ID	EXTRACT		CERTGRAPH			OPTIMA			M-CERT		AIPOC		UI/UX			INTEROP		PILOTS	
	KPI1.1	KPI1.2	KPI2.1	KPI2.2	KPI2.3	KPI3.1	KPI3.2	KPI3.3	KPI4.1	KPI4.2	KPI5.1	KPI5.2	KPI6.1	KPI6.2	KPI6.3	KPI7.1	KPI7.2	KPI8.1	KPI8.2
AI-SEC.01		X									X	X							
AMOE.01	X																		
AMOE.02	X																		
AMOE.03	X																		
AMOE.04	X																		
AMOE.05	X																		
AMOE.06	X																		
AMOE.07	X																		

Req. ID	EXTRACT		CERTGRAPH			OPTIMA			M-CERT		AIPOC		UI/UX			INTEROP		PILOTS	
	KPI1.1	KPI1.2	KPI2.1	KPI2.2	KPI2.3	KPI3.1	KPI3.2	KPI3.3	KPI4.1	KPI4.2	KPI5.1	KPI5.2	KPI6.1	KPI6.2	KPI6.3	KPI7.1	KPI7.2	KPI8.1	KPI8.2
CLDISC.01	X																		
CODYZE.01	X																		
EKNOWS.01	X																		
EKNOWS.02	X																		
EKNOWS.03	X																		
EKNOWS.04	X																		
EKNOWS.05	X																		
TWS.01																X			
TWS.02																			
TWS.03																			
TWS.04																	X		
MARI 1.0						X	X	X											
MARI 2.0						X	X	X											
MARI 3.0						X	X	X											
MARI 4.0						X	X	X											
MARI 5.0						X	X	X											
RCM.01									X	X									
RCM.02																			
RCM.03									X	X									
RCM04						X		X											
RCM.05																X	X		
RCM.06																X			
RCM.07																X	X		
RCM.08																X	X		
ORCH.01									X	X									
ORCH.02														X					
ORCH.03															X				
ORCH.04				X															
ORCH.05																			
ESTORE.01			X		X														
ESTORE.02	X																		
ASSESS.01									X	X									
ASSESS.02										X									
ASSESS.03																			

Req. ID	EXTRACT		CERTGRAPH			OPTIMA			M-CERT		AIPOC		UI/UX			INTEROP		PILOTS	
	KPI1.1	KPI1.2	KPI2.1	KPI2.2	KPI2.3	KPI3.1	KPI3.2	KPI3.3	KPI4.1	KPI4.2	KPI5.1	KPI5.2	KPI6.1	KPI6.2	KPI6.3	KPI7.1	KPI7.2	KPI8.1	KPI8.2
EVAL.01																			
EVAL.02																			

It can be seen from Table 9 that some of the KPIs are not directly addressed by any technical requirements. But this does not mean they are not covered by the EMERALD framework. In fact, they are generic KPIs that affect the whole framework and are addressed in a holistic manner. These are the KPIs in question (coloured in the table):

- KPI 6.1 (related to providing roles and workflows, develop mock-ups for the audit process): It is closely related with all the work being carried in the WP4, where an UI/UX design process with stakeholders is leading to the definition of the roles and a set of mock-ups.
- KPI 8.1, KPI 8.2 (related with pilots' implementation and validation): This aspect is being covered by the WP5, where the pilots have been designed and, in general, the whole EMERALD framework is covering them.

3.4.3 Mapping of requirements to Business Driven Requirements

In the end, the business-driven requirements (BDRs) must be implemented in the components. To ensure the technical implementation, the business-driven requirements were reviewed in collaboration with WP5 in joint workshops and mapped to technical requirements. This work assigns a list of component technical requirements to each business-driven requirement.

The alignment in Table 10is intended to show that each BDR defined by the Pilots has one or more corresponding components that implement it. In this way, a Pilot can identify the component responsible for implementing each BDR and track its coverage along the time.

A BDR with no associated functional requirements means that it is either out of scope of the EMERALD framework -as it is currently defined- or that the framework doesn't contemplate all the user needs. In the latter case, this table will serve for components designers to identify missing functionalities from the Pilots perspective, thus aligning both perspectives used for the elicitation of the functional requirements.

Table 10. Technical requirements vs Business Requirements alignment matrix.

Req. ID	Pilot 1 Ionos							Pilot 2 Cloudferro					Pilot 3 Fabasoft												Pilot 4 Caixabank						
	BDRP1.01	BDRP1.02	BDRP1.03	BDRP1.04	BDRP1.05	BDRP1.06	BDRP1.07	BDRP2.01	BDRP2.02	BDRP2.03	BDRP2.04	BDRP2.05	BDRP3.01	BDRP3.02	BDRP3.03	BDRP3.04	BDRP3.05	BDRP3.06	BDRP3.07	BDRP3.08	BDRP3.09	BDRP3.10	BDRP3.11	BDRP3.12	BDRP4.01	BDRP4.02	BDRP4.03	BDRP4.04	BDRP4.05	BDRP4.06	BDRP4.07
AI-SEC.01														X												X					
AMOE.01					X																		X			X					
AMOE.02					X																		X			X					
AMOE.03																							X								

Req. ID	Pilot 1 Ionos							Pilot 2 Cloudferro					Pilot 3 Fabasoft												Pilot 4 Caixabank						
	BDRP1.01	BDRP1.02	BDRP1.03	BDRP1.04	BDRP1.05	BDRP1.06	BDRP1.07	BDRP2.01	BDRP2.02	BDRP2.03	BDRP2.04	BDRP2.05	BDRP3.01	BDRP3.02	BDRP3.03	BDRP3.04	BDRP3.05	BDRP3.06	BDRP3.07	BDRP3.08	BDRP3.09	BDRP3.10	BDRP3.11	BDRP3.12	BDRP4.01	BDRP4.02	BDRP4.03	BDRP4.04	BDRP4.05	BDRP4.06	BDRP4.07
AMOE.04					X																		X								
AMOE.05																										X					
AMOE.06																										X					
AMOE.07																							X								
CLDISC.01								X																							
CODYZE.01						X																				X					X
EKNOWS.01																										X					
EKNOWS.02																															
EKNOWS.03																															
EKNOWS.04																															
EKNOWS.05						X																									
TWS.01		X								X										X						X	X				
TWS.02		X								X										X						X	X				
TWS.03											X		X															X			
TWS.04		X								X																X	X				
MARI 1.0			X							X							X			X					X	X	X				
MARI 2.0			X						X	X		X				X	X	X		X					X						
MARI 3.0																															
MARI 4.0																X		X													
MARI 5.0			X						X							X	X	X		X					X						
RCM.01												X					X														
RCM.02																										X				X	
RCM.03																	X														
RCM04									X							X	X														
RCM.05				X														X						X	X						
RCM.06				X																				X							
RCM.07				X														X							X						
RCM.08				X																		X									
ORCH.01	X									X										X		X					X	X		X	
ORCH.02											X		X													X	X	X	X	X	
ORCH.03																													X	X	
ORCH.04	X																											X	X	X	

Req. ID	Pilot 1 Ionos							Pilot 2 Cloudferro					Pilot 3 Fabasoft												Pilot 4 Caixabank						
	BDRP1.01	BDRP1.02	BDRP1.03	BDRP1.04	BDRP1.05	BDRP1.06	BDRP1.07	BDRP2.01	BDRP2.02	BDRP2.03	BDRP2.04	BDRP2.05	BDRP3.01	BDRP3.02	BDRP3.03	BDRP3.04	BDRP3.05	BDRP3.06	BDRP3.07	BDRP3.08	BDRP3.09	BDRP3.10	BDRP3.11	BDRP3.12	BDRP4.01	BDRP4.02	BDRP4.03	BDRP4.04	BDRP4.05	BDRP4.06	BDRP4.07
ORCH.05	X																									X	X	X	X	X	
ESTORE.01																					X					X			X		
ESTORE.02															X														X		
ASSESS.01																											X				
ASSESS.02																	X														
ASSESS.03										X										X							X				
EVAL.01	X																										X				
EVAL.02	X																														

As mentioned above, each BDR to be implemented should be related to at least one component. Otherwise, it would mean that no component is implementing such requirement. According to the Table 10, the BDRs that fall into this category are the following:

- *BDRP1.07 - Intuitive User Experience for Compliance Monitoring*: this requirement is addressed by all the UI/UX requirements developed in WP4.
- *BDRP3.07 - Enhance current audit process*: It is a very generic requirement that must be addressed by the whole EMERALD platform, as all components are involved in the improving the audit process.

3.4.4 Prioritization and current status

Table 11 depicts the status of the functional requirements foreseen for M12 (at milestone *MS2: Components V1*), the due date of this deliverable. For a complete table with the status of all requirements, view the *APPENDIX A: Current status of requirements*.

Table 11. Requirements prioritization matrix

Req. ID	Title	Priority	Timeline	Status
AI-SEC.01	The extractor tool includes selected criteria	MUST	M12 (C-v1)	35%
AMOE.01	Upload PDF document	MUST	M12 (C-v1)	90%
AMOE.04	Compare results from multiple documents	SHOULD	M12 (C-v1)	70%
TWS.01	Provide integrity proof of evidence	MUST	M12 (C-v1)	75%
TWS.02	Provide integrity proof of assessment results	MUST	M12 (C-v1)	75%
RCM.01	Multi-schema support	MUST	M12 (C-v1)	90%
RCM.02	Accessible by the rest of components	MUST	M12 (C-v1)	100%
RCM.03	Include metrics for all schemas supported	MUST	M12 (C-v1)	30%
RCM.06	Import/export of security schemes in CSV format	COULD	M12 (C-v1)	60%

Req. ID	Title	Priority	Timeline	Status
ORCH.02	REST API Gateway for UI	MUST	M12 (C-v1)	15%

3.5 Requirements Summary Dashboard

Table 12 shows a summary of requirements by component, with their status -in a broad vision divided in not started, partially implemented and fully implemented- at the moment of writing.

Table 12. Summary table of requirements status at M12 (by component)

Component	Not started	Partially implemented	Fully implemented	TOTAL
AI-SEC	0	1	0	1
AMOE	4	3	0	7
Discovery	0	1	0	1
Codyze	0	1	0	1
eKnows	1	4	0	5
TWS	0	4	0	4
MARI	0	5	0	5
RCM	1	6	1	8
Evidence Store	0	2	0	2
Orchestrator	3	2	0	5
Assessment	1	2	0	3
Evaluation	1	1	0	2
NFR (WP1)	0	7	1	8
TOTAL	**11**	**39**	**2**	**52**

It can be observed that, because of the different ranges of functionality of each component, the requirements are not equally distributed among the components (see Figure 4). It is also the case that not all components have yet the same level of definition. In this respect, the components with the most requirements are *RCM* (with 8), *AMOE* (with 7) and *MARI*, *Orchestrator* and *eknows* (with 5 each).

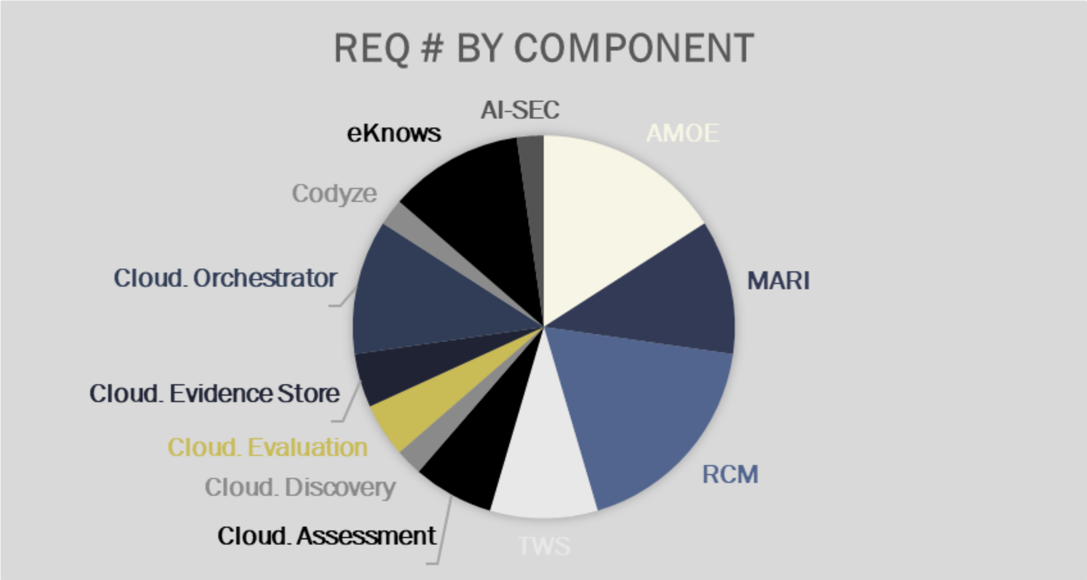


Figure 4. Number of requirements per component

Regarding the status of the requirements at M12 (see Figure 5), most of them are in a work in progress status (32 out of 52); the not-started requirements are half of the started ones (16 out of 52); and few requirements are already fully implemented (4 of 52).

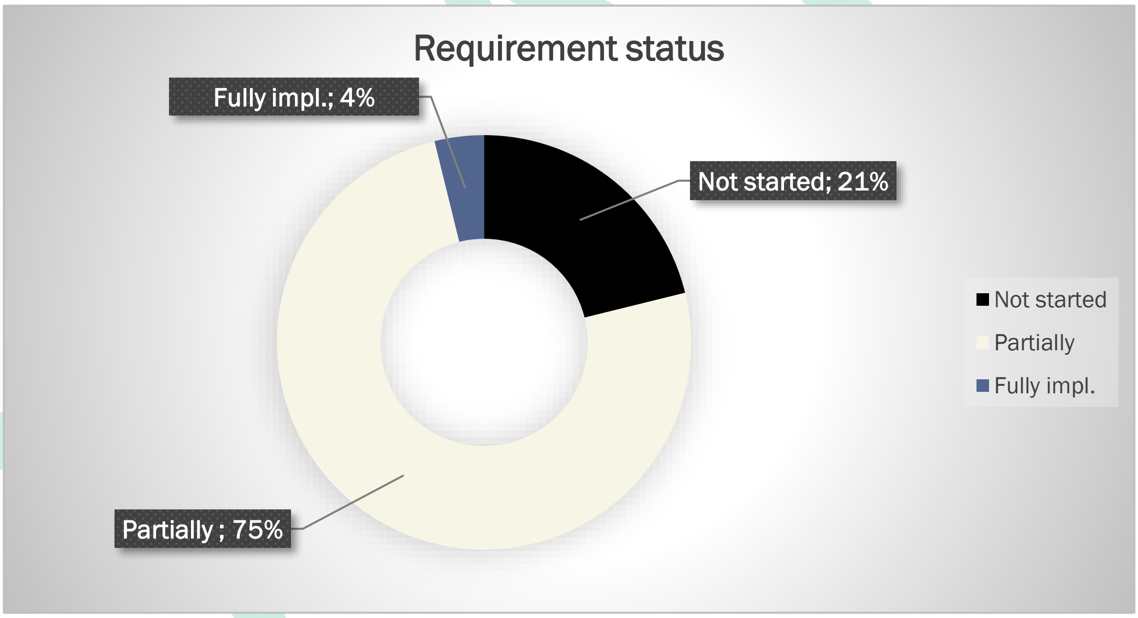


Figure 5. Requirement status

Figure 6 shows the status of requirements by component. Logically, the same pattern that in the overall view can be observed, i.e., all the components have a majority of partially implemented requirements, with some requirements nor yet started and only a few completed requirements.

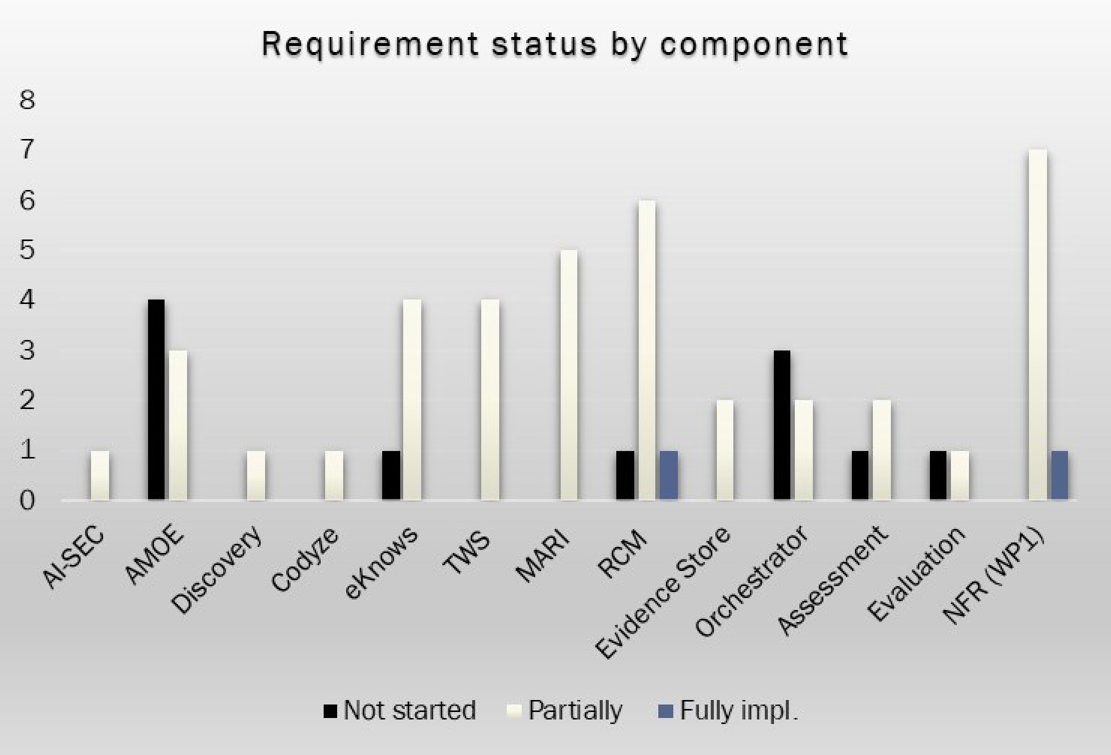


Figure 6. Requirement status per component

Finally, let's have a look to the coverage of the requirement sets to the different pilots. Table 13 shows the number of requirements for each component that cover some aspect of each pilot (a pilot requirement).

We can see that the most covered pilot is Pilot 4, with 50 requirements, followed by Pilot 3 (38), Pilot 2 (17) and Pilot 1 (16). The colour shows, for each pilot, which component contributes the most (red), with the intensity decreasing as the contribution of the component to the pilot decreases.

Table 13. GENERAL VIEW: Components vs Pilot

Component	Pilot 1	Pilot 2	Pilot 3	Pilot 4	TOTAL
AMOE	3	0	5	4	12
MARI	3	5	12	5	25
RCM	4	2	7	4	17
TWS	3	4	3	7	17
Cloud. Assessment	0	1	2	2	5
Cloud. Discovery	0	1	0	0	1
Cloud. Evaluation	2	0	0	1	3
Cloud. Evidence Store	0	0	2	3	5
Cloud. Orchestrator	3	2	3	18	26
Codyze	1	0	0	1	2
eKnows	1	0	0	1	2
AI-SEC	0	0	1	1	2
NFR	1	0	0	6	7

4 EMERALD Framework detailed view

This section describes the architecture of the EMERALD CaaS framework. It provides a succinct description of the components that make up the EMERALD framework, their workflows, implemented interfaces, and sequence diagrams.

4.1 Data model

The EMERALD data model was defined in D1.1 [1], that describes the different data classes used by the components, and the connections within and between components. The data model is useful mainly for the developers of the EMERALD framework in order to construct the software classes to manage the required data structures.

The data model for the whole EMERALD framework is shown in Figure 7, where each component is represented in a box, that includes inside the data structures it handles. The background colour of the box denotes the project work package to which the component pertains. Thus, Evidence Collection components (WP2) are coloured in orange, whereas WP3 components are coloured in teal.

The EMERALD project uses some of the components that were part of the MEDINA data model – such as the **Evidence Store**, the **Orchestrator**, the **Repository of Controls and Metrics (RCM)** and the **Trustworthiness System**.

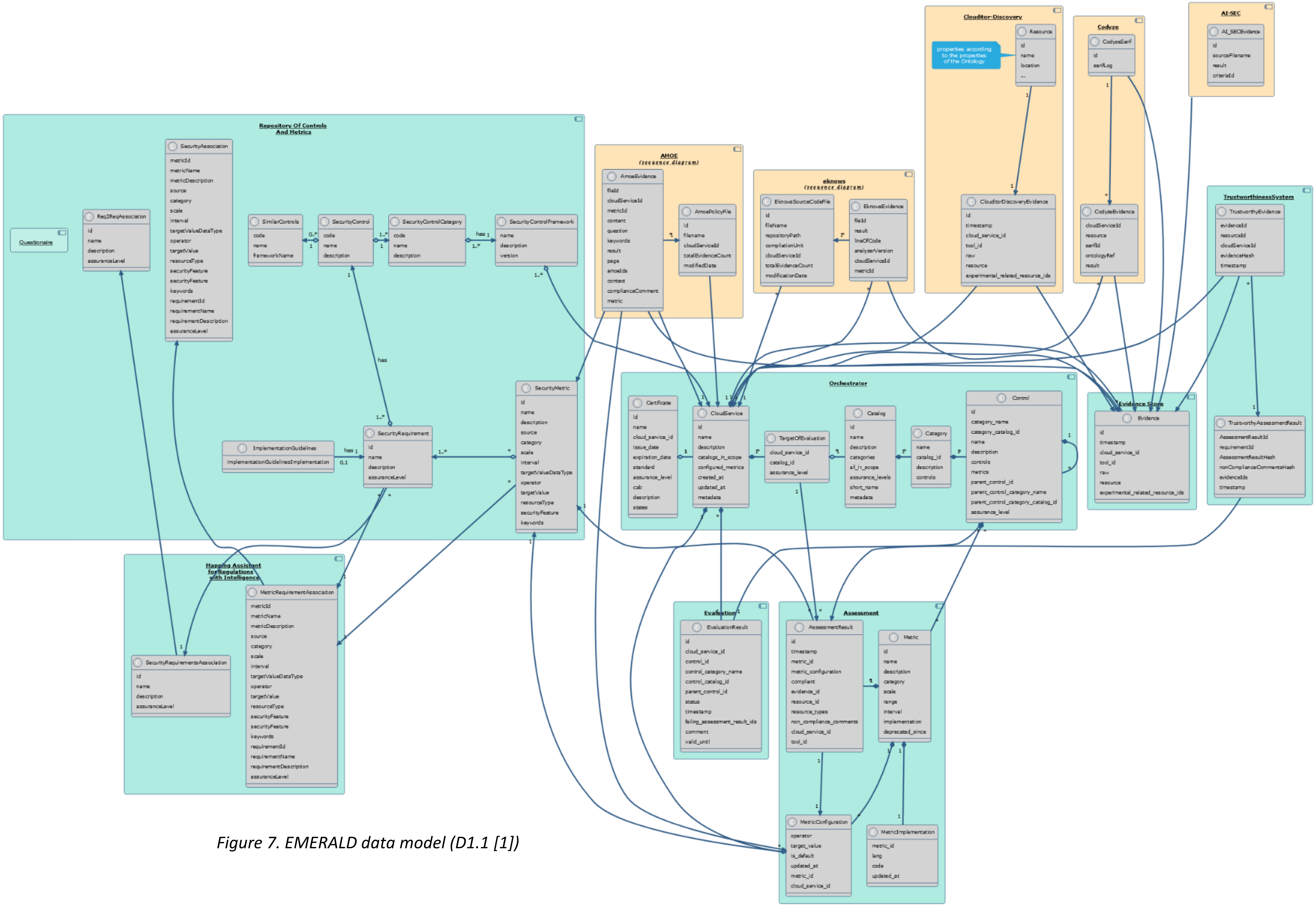


Figure 7. EMERALD data model (D1.1 [1])

The main data structures used by EMERALD components are listed in the following:

AMOE

- *AmoePolicyFile:* serves as an internal representation of the uploaded file, which can be linked to a *Cloud Service* via it's id.

Clouditor-Discovery

- *Resource:* stores any cloud resource, also used in the EMERALD Graph Ontology.

Codyze

- *CodyzeSarif:* where the generated analysis report in SARIF[9] is stored. Moreover, *Codyze* processes the findings in the SARIF report into evidence for the EMERALD framework.

eknows

- *EknowsSourceCodeFile* serves as an internal representation of the source code file to be analysed.

MARI

- *SecurityRequirementsAssociation:* stores association among requirements or controls, as a result of are the *MARI* processing.
- *MetricRequirementAssociation:* stores association among metrics and requirements or controls.

RCM

- *SecurityControlFramework*: defines the standard schema (e.g., EUCS)
- *SecurityCategory*: defines a category of the schema.
- *SecurityControl:* defines a control of the category and can have a list of sub-controls.
- *SecurityRequirement:* defines a requirement inside a control.
- *SimilarControls:* supports mapping among controls of different schemes.
- *ImplementationGuidelines***:** help the user with the implementation of the requirements.
- *SecurityMetric:* defines a metric, what to measure to assess the collected evidence.

Orchestrator

- *CloudService:* holds the logical representation of a single service.
- *TargetOfEvaluation:* combines a cloud service with one dedicated security catalogue to produce a *Certificate.*
- *Certificate:* representing different states of a certificate.
- *Control:* representation of either a control, requirement or objective, as every security scheme uses different names.
- *Catalogue:* represents the security schema.
- *Category:* represents a category of controls in the schema.

Evidence Store

- *Evidence:* holds the necessary information regarding the collected evidence, including the *timestamp* describing when the evidence was created, the *Cloud Service* the evidence

[9] Static Analysis Results Interchange Format (SARIF), https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html

belongs to, the ID of the evidence collector tool that created the evidence and the resource properties.

Assessment

- *MetricConfiguration:* contains the target value and the operator used in the assessment.
- *AssessmentResult:* contains the result of the assessment, including information about the evidence and the metric.

Evaluation

- *EvaluationResult:* maps the measurements of individual metrics and combines them according to the mapping of a metric to a *Control*. Includes a timestamp and a status, and also related information like control or cloud service.

As mentioned before, we only provide in this document a general view of the data model, because specific details of the data models used by each component has been provided in deliverable *D1.1 – Data Modelling and interaction mechanisms* [1]. For more detailed information, please go to this deliverable.

4.2 Component description (components cards & sequence diagrams)

This section contains a description of the EMERALD components. It covers the evidence extraction tools —that extract, store and assess the evidence— and the tools that provide and assist with the management of the security schemes and metrics.

Please note that the data-oriented point of view of each component was already covered in D1.1 [1], so this document will not repeat it, but has only presented an overview of the data model for completeness in Section 4.1.

4.2.1 Evidence Collectors

4.2.1.1 AI-SEC

<table>
<tr><td><b>Component Name</b></td><td>AI-SEC</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:
• Extracting evidence by given machine learning model, data, criteria
• Providing evidence to the Orchestrator to further assessment</td></tr>
<tr><td><b>Sub-components Description</b></td><td>Currently no division in subcomponents is planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>Management</td><td>This interface handles operations related to the management of ML models, such as uploading, downloading, updating, and deleting models.</td><td>Rest API</td></tr>
<tr><td>Inference</td><td>This interface enables the execution of the model to make predictions or perform inference. The REST API here would allow</td><td>Rest API</td></tr>
</table></td></tr>
</table>

	users to send data for inference and receive predictions in return.	
Monitoring	This interface tracks the performance of the model over time, monitors inference requests, and logs errors.	Rest API

Interaction with other components	- Evidence Store ○ Submit evidence to be stored
Relevant sequence diagram/s	See section 4.2.1.1.1
Requirements Mapping	List of requirements covered by this component: - AI-SEC.01: the extractor tool includes selected criteria
Technology used	We use some Open Source to extract evidence, such as CLEVER[10] and LIME[11]
Related KR	KR5
WP and task	WP2 – T2.4
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.1.1.1 Sequence diagram

Figure 9 shows the sequence diagram of the *AI-SEC* component. The user configures the *AI-SEC* parameters in the CI/CD pipeline, which triggers the *AI-SEC* analysis to start. Evidence is then gathered and mapped in the ontology and sent to *the Evidence Store*.

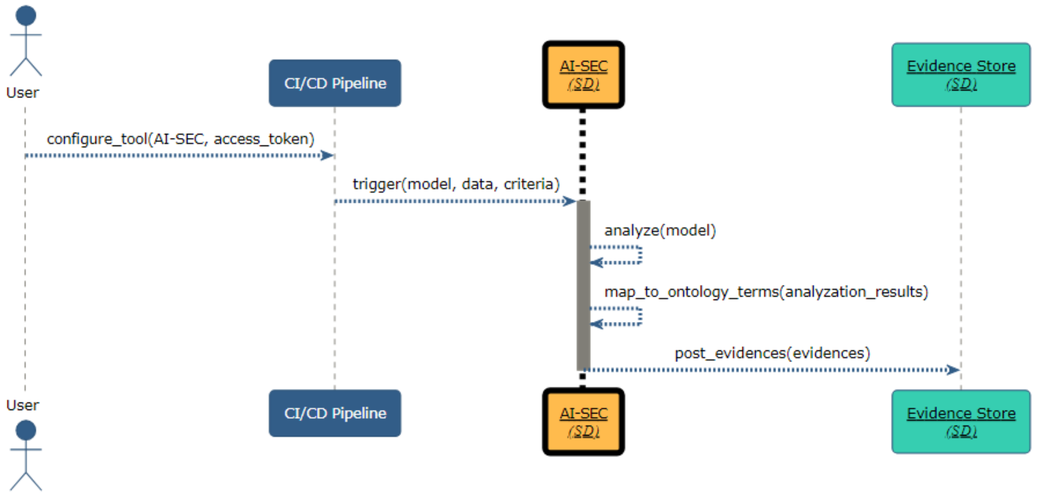

User
CI/CD Pipeline
AI-SEC (SD)
Evidence Store (SD)
configure_tool(AI-SEC, access_token)
trigger(model, data, criteria)
analyze(model)
map_to_ontology_terms(analyzation_results)
post_evidences(evidences)
User
CI/CD Pipeline
AI-SEC (SD)
Evidence Store (SD)

Figure 9. AI-SEC sequence diagram

[10] https://github.com/IBM/CLEVER-Robustness-Score
[11] https://github.com/marcotcr/lime

4.2.1.2 AMOE

<table>
<tr><td><b>Component Name</b></td><td>Assessment and Management of Organisational Evidence (AMOE)</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:<ul><li>Gathering and processing organizational evidence</li><li>Providing evidence to the <i>Evidence Store</i> and <i>Assessment</i> components</li></ul></td></tr>
<tr><td><b>Sub-components Description</b></td><td>Organizational evidence is collected by applying NLP and organisational metrics to an uploaded document. The processing part transforms this evidence in the form of technical evidence. This transformed evidence is then provided to the security assessment of the Clouditor which can handle such technical evidence.</td></tr>
<tr><td><b>Main logical Interfaces</b></td><td><table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>UI</td><td>GUI to
Upload documents
Retrieve evidence
Set assessment results
Submit/forward assessment results</td><td>webservice</td></tr>
<tr><td>API</td><td>Upload documents
Retrieve evidence
Set assessment results
Submit/forward assessment results</td><td>REST</td></tr>
</table></td></tr>
<tr><td><b>Interaction with other components</b></td><td><table>
<tr><th>Interfacing Component</th><th>Interface Description</th></tr>
<tr><td>Evidence Store</td><td>Send collected evidence</td></tr>
<tr><td>Orchestrator</td><td>Retrieve metric configurations</td></tr>
<tr><td>Repository of Controls and Metrics</td><td>Retrieve metrics and requirements as needed</td></tr>
</table></td></tr>
<tr><td><b>Relevant sequence diagram/s</b></td><td>See section 4.2.1.2.1</td></tr>
<tr><td><b>Requirements Mapping</b></td><td>List of requirements covered by this component:<ul><li>AMOE.01: Upload PDF document</li><li>AMOE.02: Provision of extracted evidence to EvidenceStore (Orchestrator/Clouditor)</li><li>AMOE.03: Refine evidence extraction approach</li><li>AMOE.04: Compare results from multiple documents</li><li>AMOE.05: Select metrics per document</li><li>AMOE.06: Classify document, select respective metrics (optional)</li><li>AMOE.07: Metric states</li></ul></td></tr>
<tr><td><b>Technology used</b></td><td>Python</td></tr>
<tr><td><b>WP and task</b></td><td>WP2: T2.3</td></tr>
<tr><td><b>Related KR</b></td><td>KR1, KR2, KR8</td></tr>
<tr><td><b>License</b></td><td>Apache 2.0</td></tr>
</table>

Partner	FABA

4.2.1.2.1 Sequence diagram

Figure 10 shows the sequence diagram of the *AMOE* component. *AMOE* extracts evidence which target specific parts of policy documents. After the extraction process, the evidence can be inspected in a GUI that comes with *AMOE* or retrieved via the API.

AMOE works with metrics from the *RCM* and accesses the target values from the *Orchestrator* API. Files are uploaded by the Compliance Manager, and then processed to get the evidence. Once the evidence is confirmed by the Internal Auditor, it can be forwarded to the *Evidence Store. AMOE* provides its functionalities to the *EMERALD UI* via an API.

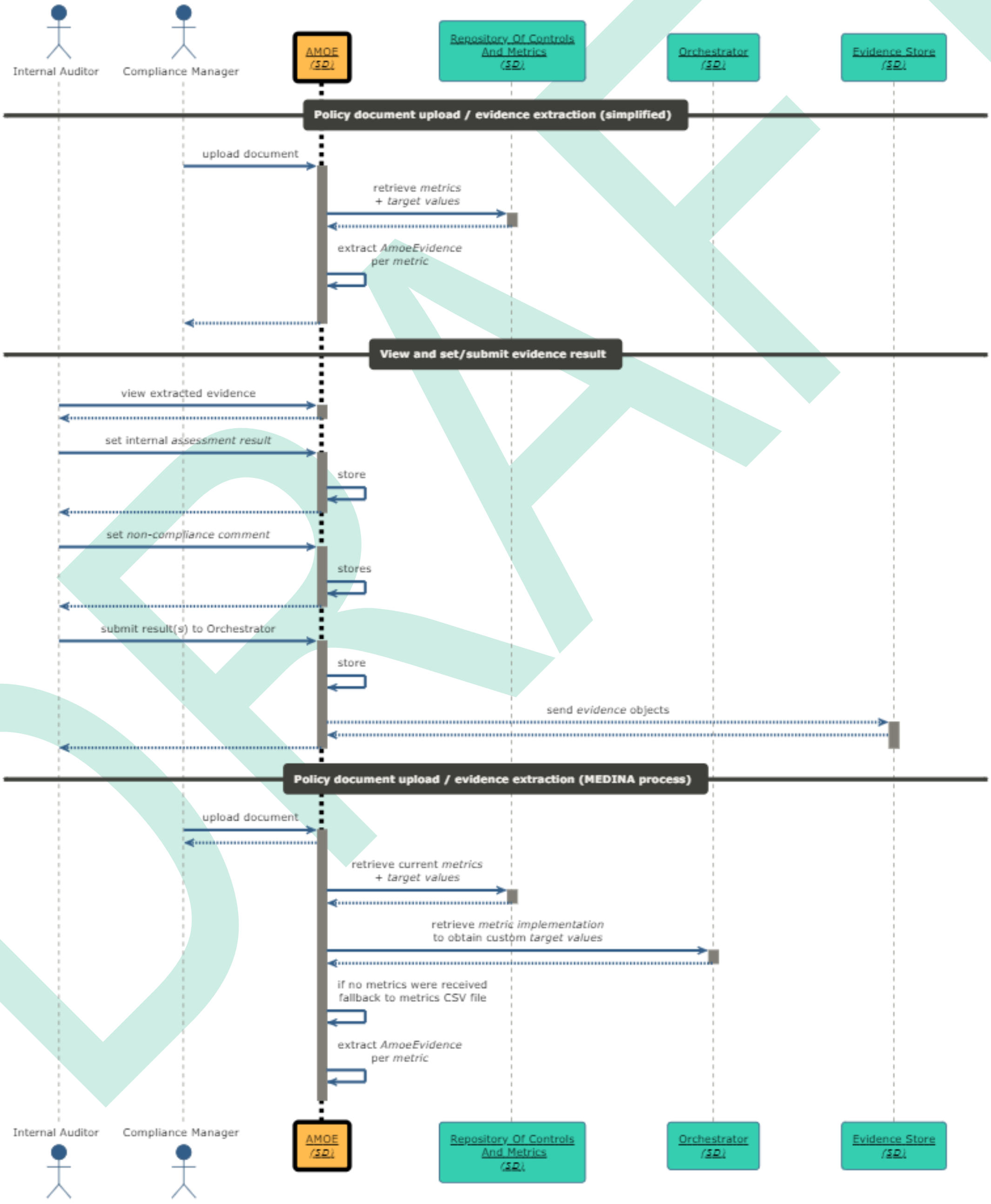

Internal Auditor
Compliance Manager
AMOE (SD)
Repository Of Controls And Metrics (SD)
Orchestrator (SD)
Evidence Store (SD)
Policy document upload / evidence extraction (simplified)
upload document
retrieve metrics + target values
extract AmoeEvidence per metric
View and set/submit evidence result
view extracted evidence
set internal assessment result
store
set non-compliance comment
stores
submit result(s) to Orchestrator
store
send evidence objects
Policy document upload / evidence extraction (MEDINA process)
upload document
retrieve current metrics + target values
retrieve metric implementation to obtain custom target values
if no metrics were received fallback to metrics CSV file
extract AmoeEvidence per metric
Internal Auditor
Compliance Manager
AMOE (SD)
Repository Of Controls And Metrics (SD)
Orchestrator (SD)
Evidence Store (SD)

Figure 10. AMOE sequence diagram

4.2.1.3 Clouditor-Discovery

<table>
<tr><td><b>Component Name</b></td><td>Clouditor-Discovery</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:<ul><li>Extracts cloud configurations for different Cloud resources (e.g., Virtual Machine, Object Storage, Network Interface) from several Cloud providers (e.g., Azure) via API calls.</li><li>Stores the extracted information in the EMERALD evidence format in the <i>Evidence Store</i> component.</li></ul></td></tr>
<tr><td><b>Sub-components Description</b></td><td>Currently no division in subcomponents planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><table><tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr><tr><td>CLI</td><td>A CLI is available</td><td>Cobra[12]/Viper[13]</td></tr><tr><td>API</td><td>The following endpoints are available:<ul><li>Start to start the discovery.</li><li>ListResources lists discovered resources.</li></ul></td><td>REST/gRPC</td></tr></table></td></tr>
<tr><td><b>Interaction with other components</b></td><td><ul><li><i>Evidence Store</i>: Submit evidence to the Evidence Store</li><li><i>Orchestrator</i>: Registers the Clouditor-Discovery component in the Orchestrator (not yet implemented, to be discussed).</li></ul></td></tr>
<tr><td><b>Relevant sequence diagram/s</b></td><td>See section 4.2.1.3.1</td></tr>
<tr><td><b>Requirements Mapping</b></td><td>List of requirements covered by this component:<ul><li>CLDISC.01: Discovery of security features of infrastructure components</li></ul></td></tr>
<tr><td><b>Technology used</b></td><td>Go[14], gRPC[15]</td></tr>
<tr><td><b>Related KR</b></td><td>KR1_EXTRACT</td></tr>
<tr><td><b>WP and task</b></td><td>WP2 – T2.5</td></tr>
<tr><td><b>License</b></td><td>Apache-2.0</td></tr>
<tr><td><b>Partner</b></td><td>Fraunhofer AISEC</td></tr>
</table>

4.2.1.3.1 Sequence diagram

Figure 11 shows the sequence diagram of the *Clouditor-Discovery* component. *Clouditor-Discovery* identifies various cloud resources and discovers security-relevant configurations, such as encryption in use, restricted ports, etc., to enhance security compliance.

It is registered in the system by an *AuthorizedEntity*, and then registers itself in the O*rchestrator*. Once started, it continuously retrieves runtime information from the cloud resources, and stores them in the *EvidenceStore.*

[12] https://github.com/spf13/cobra
[13] https://github.com/spf13/viper
[14] https://go.dev/
[15] https://grpc.io/

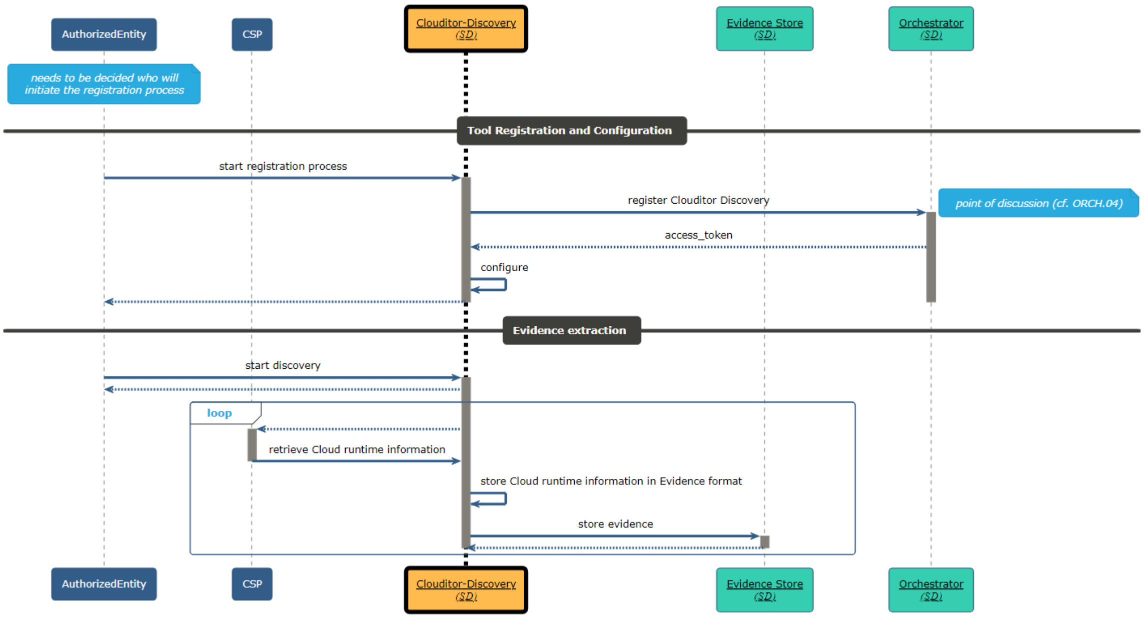

AuthorizedEntity
CSP
Clouditor-Discovery (SD)
Evidence Store (SD)
Orchestrator (SD)
needs to be decided who will initiate the registration process
Tool Registration and Configuration
start registration process
register Clouditor Discovery
point of discussion (cf. ORCH.04)
access_token
configure
Evidence extraction
start discovery
loop
retrieve Cloud runtime information
store Cloud runtime information in Evidence format
store evidence
AuthorizedEntity
CSP
Clouditor-Discovery (SD)
Evidence Store (SD)
Orchestrator (SD)

Figure 11. Clouditor-Discovery sequence diagram

4.2.1.4 Codyze

<table>
<tr><td><b>Component Name</b></td><td>Codyze</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:
<ul>
<li>Scans source code for insecure implementations of security-relevant features (e.g., transport encryption, logging, authentication & authorisation, etc.)</li>
<li>Analyse interactions between cloud service components from infrastructure-as-code (e.g., What cloud resources are consumed?, Are interactions secure?, Are used resources up-to-date?, etc.)</li>
<li>Analyse development processes (e.g., Are secure development processes followed?, Is the provenance of source code guaranteed?, What measures are taken to secure the development pipeline?, etc.)</li>
</ul></td></tr>
<tr><td><b>Sub-components Description</b></td><td>Currently no division in subcomponents planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td>
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>CLI</td><td>A CLI incl. configuration file to configure Codyze and set execution/analysis parameters.</td><td>Kotlin Clikt library[16]</td></tr>
</table>
</td></tr>
<tr><td><b>Interaction with other components</b></td><td>
<ul>
<li><i>Orchestrator</i>
<ul><li>Request information on cloud service to be analysed</li></ul></li>
<li><i>Evidence Store</i>
<ul><li>Submit evidence to be stored</li></ul></li>
</ul></td></tr>
</table>

[16] https://ajalt.github.io/clikt/

Relevant sequence diagram/s	See section 4.2.1.4.1
Requirements Mapping	List of requirements covered by this component: • CODYZE.01: Extraction of security features from source code
Technology used	Kotlin[17]
Related KR	KR1
WP and task	WP2 – T2.2
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.1.4.1 Sequence diagram

Figure 12 shows the sequence diagram of the *Codyze* component. *Codyze* provides evidence extraction from source code of cloud services. It analyses and generates evidence results that indicate if code segments are compliant or non-compliant to specified requirements. These evidence results are submitted to the E*vidence Store* for storage and further processing.

As in the case of *AI-SEC*, it is recommended to run it as part of a CI/CD pipeline, that prevents the deployment of non-compliant services and application. For that, some initial configuration is needed.

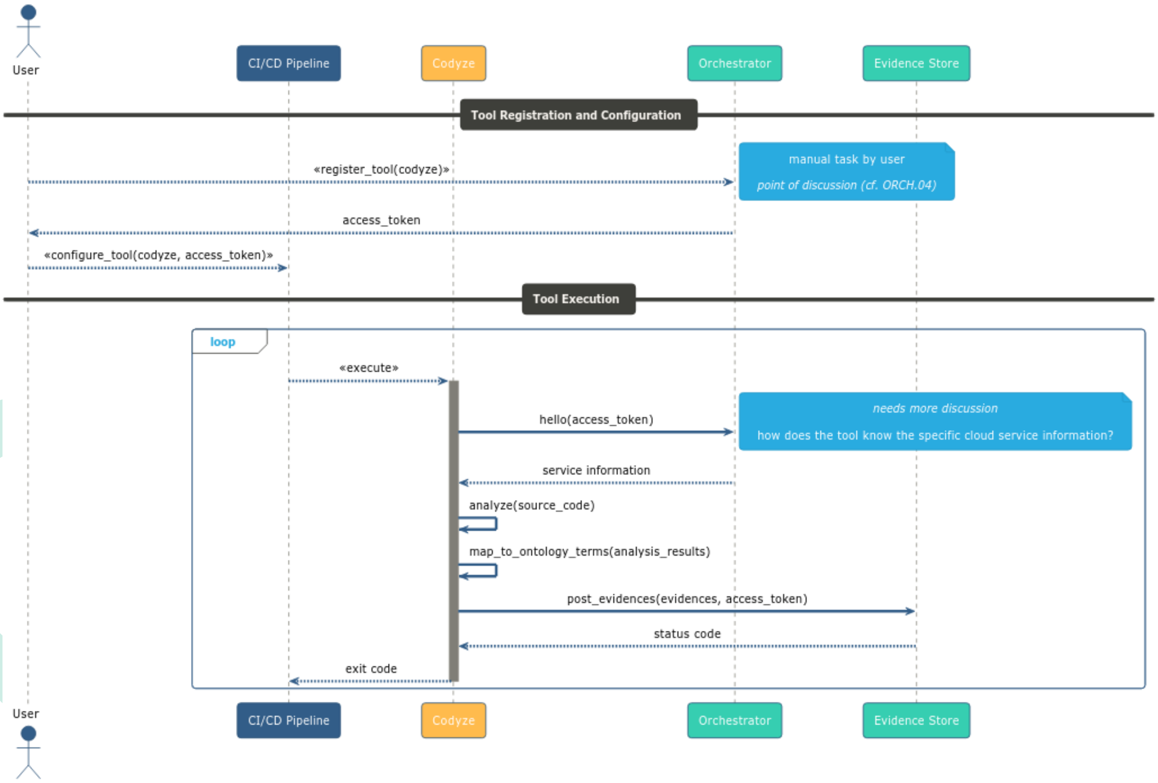
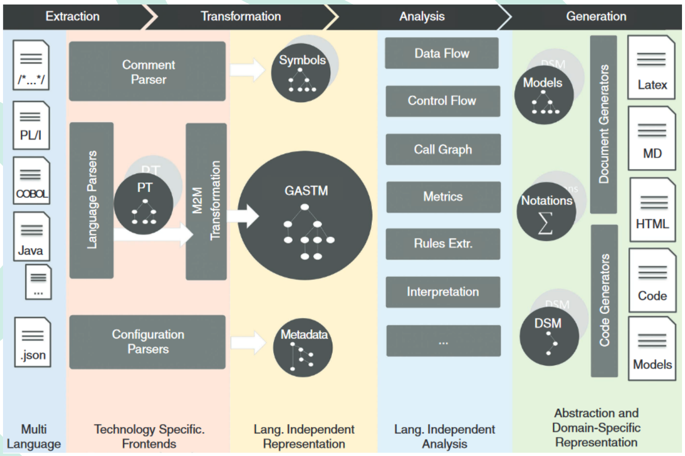


Figure 12. Codyze sequence diagram

[17] https://kotlinlang.org/

4.2.1.5 eknows

<table>
<tr><td><b>Component Name</b></td><td>eknows</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:
• Static code analysis.
• Language-independent frontends (currently >16 programming languages, including Java, Python, Cobol, C++, etc.).
• Rapid development platform for software tools such as documentation generators and tools for reverse engineering and code visualization.
• Extraction of business rules from code.</td></tr>
<tr><td><b>Sub-components Description</b></td><td><i>eknows</i> is a Java-based software platform to build reverse engineering tools and documentation generators. The platform provides a modular extensible set of software components, which facilitate the rapid development of tools in program comprehension, documentation generation, and software reverse engineering. Support for multiple programming languages in terms of language-specific extraction components and language-independent analysis is a key feature of the platform.
The platform (see Figure 13) provides reusable components that facilitate (i) language parsing (extraction), (ii) transformation of source code into a generic abstract syntax tree (GASTM), (iii) structural and behavioural analysis of software, and (iv) reporting and visualization of analysis results.


<i>Figure 13. Overview of eknows platform components</i>
Tools built on top of <i>eknows</i> integrate required software components as-is and add functionality required for a specific use case.</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>Java API</td><td><i>eknows</i> can be added as a set of Java libraries (<i>eknows-core, eknows-frontends, eknows-analysis</i>, etc.) to call its components.</td><td>Java</td></tr>
<tr><td>REST (maybe)</td><td>The analyzation of source code files can be triggered via a REST endpoint.</td><td>HTTP / REST</td></tr>
</table></td></tr>
</table>

CLI	The analyzation of source code files can be triggered via a command line interface.	stdin/stdout

	Note: REST interface does not exist yet, however, if needed, it will be developed within EMERALD.
Interaction with other components	• *Evidence Store*: Sends (raw) evidence. • *CI/CD Pipeline:* Starts analyzation of source code files by calling a trigger provided by *eknows*.
Relevant sequence diagram/s	See section 4.2.1.5.1
Requirements Mapping	The requirements covered by this component are: • EKNOWS.01 – Integration into existing systems • EKNOWS.02 – Resilience while analysing erroneous code • EKNOWS.03 – Multi-language support • EKNOWS.04 – Support EMERALD evidence format • EKNOWS.05 – Static code analysis
Technology used	Java Ecosystem
Related KR	KR1 EXTRACT
WP and task	WP2 – T2.2
License	*eknows*-core, reused frontends and reused analyses eknows Binary Usage Software License *eknows* extractor Apache License, Version 2.0
Partner	SCCH

4.2.1.5.1 Sequence diagram

Figure 14 shows the sequence diagram of the *eknows* component. *eknows* supports the creation of evidence extraction functions by reusing prefabricated parsing, analysis, and generation modules, with the mission to verify if application source code complies to security requirements.

eknows can be integrated into CI/CD pipelines by using the binary distribution. Findings are generated as console output. This output will be submitted to the *Evidence Store* of the EMERALD framework in the format of the *CertGraph* ontology.

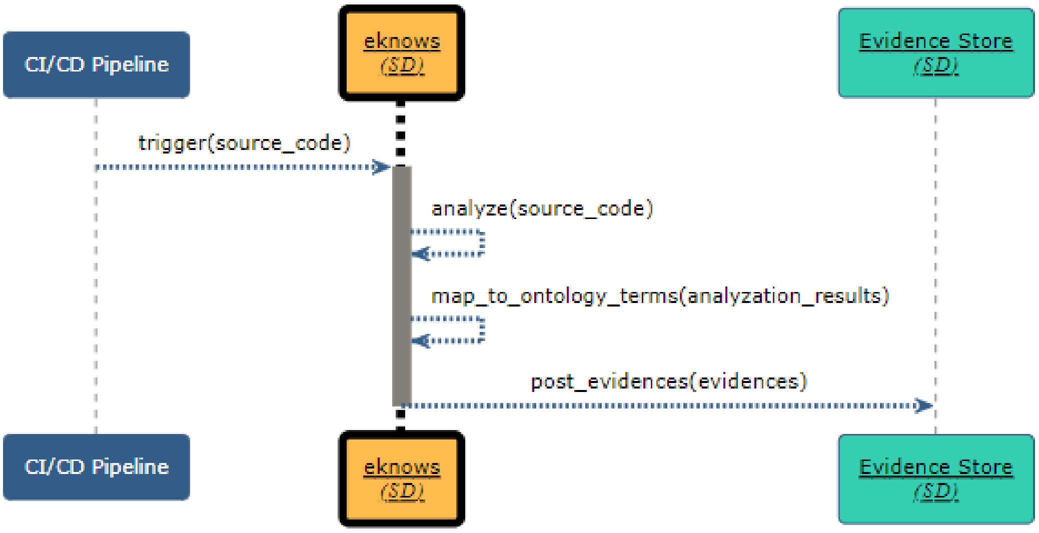


Figure 14. eknows sequence diagram

4.2.2 TWS – Trustworthiness System

Component Name	Trustworthiness System (TWS)
Main functionalities	The component provides the following functionalities: • Maintains an improved audit trail of evidence and assessment results. • Provides a manual and automatic way of verification of evidence and assessment results integrity. • Provides a record of information on a verifiable way (verification). • Provides a record of information on a permanent way (traceability). • Guarantees resistance to modification of stored data (integrity).
Sub-components Description	**Blockchain network**, use of a real implementation of a Blockchain network. EBSI will be considered as the first option for the deployment. **Blockchain client**, for providing the information (evidence/assessment results) to be saved on the Blockchain. **Smart contract**, deployed on the Blockchain network, for information (evidence/assessment results) writing and reading operations as well as events generation indicating the provision of new information. **Viewer tool**, for subscription to the Blockchain based events and notification to the different viewer clients. **Graphical viewer client**, for gathering and showing all the information saved on the Blockchain (and be able to manually verify it, without needing any interaction with the Blockchain). **Automatic verification service**, for evidence and assessment results integrity automatic check to be integrated in the GUI.

<table>
<tr><td><b>Main logical Interfaces offered</b></td><td>
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>Blockchain client</td><td>It provides: i) the required evidence and assessment results to be saved on the Blockchain, and ii) a way to obtain or check the evidence and assessment results saved on the Blockchain.</td><td>REST API</td></tr>
<tr><td>Graphical Viewer Client</td><td>It provides a GUI to manually check evidence and assessment results saved on the Blockchain.</td><td>Web</td></tr>
<tr><td>Automatic Verification Service</td><td>It provides a GUI for automatic verification of the integrity of evidence and assessment results.</td><td>REST API</td></tr>
</table>
</td></tr>
<tr><td><b>Interaction with other components</b></td><td>
<table>
<tr><th>Interfacing Component</th><th>Interface Description</th></tr>
<tr><td><i>Assessment</i></td><td>The Assessment will provide (and check, if needed) the information (evidence/assessment results) to be saved on the Blockchain by means of the Blockchain client interface.</td></tr>
<tr><td><i>EmeraldUI</i></td><td>The automatic verification service will provide the integrity verification information to the <i>EmeraldUI</i> to be shown to the EMERALD users.</td></tr>
<tr><td>Auditors</td><td>The auditors will check the information saved on the Blockchain by means of the graphical viewer client interface (manual way) or the automatic verification service interface (automatic way).</td></tr>
</table>
</td></tr>
<tr><td><b>Relevant sequence diagram/s</b></td><td>See section 4.2.2.1</td></tr>
<tr><td><b>Requirements Mapping</b></td><td>• TWS.01: Provide integrity proof of evidence
• TWS.02: Provide integrity proof of assessment results
• TWS.03: Provide access through REST API or graphical interface
• TWS.04: Use a general purpose public-private Blockchain network</td></tr>
<tr><td><b>Technology used</b></td><td>Solidity, NodeJS, React, EBSI</td></tr>
<tr><td><b>Related KR</b></td><td>KR7: INTEROP – Interoperable assessment, evidence and catalogue data</td></tr>
<tr><td><b>WP and task</b></td><td>WP3 – T3.5</td></tr>
<tr><td><b>License</b></td><td>Proprietary</td></tr>
<tr><td><b>Partner</b></td><td>TECNALIA</td></tr>
</table>

4.2.2.1 Sequence diagram

4.2.2.1.1 System recording

Figure 15 shows the sequence diagram of the *TWS Recording* component. *TWS Recording* receives from the *Assessment* component the information related to evidence and assessment results to be recorded in the Blockchain. Once this is done, the automatic verification service will be able to validate its integrity.

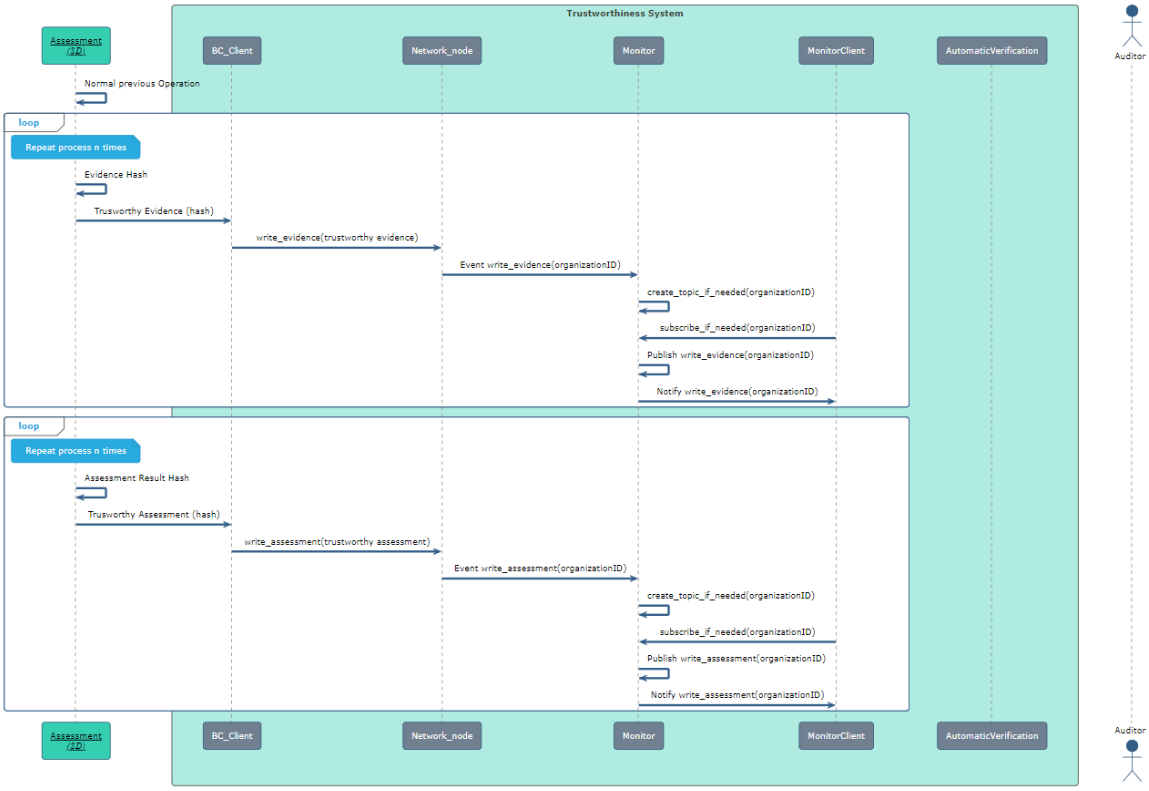

Trustworthiness System
Assessment (SD)
BC_Client
Network_node
Monitor
MonitorClient
AutomaticVerification
Auditor
Normal previous Operation
loop
Repeat process n times
Evidence Hash
Trusworthy Evidence (hash)
write_evidence(trustworthy evidence)
Event write_evidence(organizationID)
create_topic_if_needed(organizationID)
subscribe_if_needed(organizationID)
Publish write_evidence(organizationID)
Notify write_evidence(organizationID)
loop
Repeat process n times
Assessment Result Hash
Trusworthy Assessment (hash)
write_assessment(trustworthy assessment)
Event write_assessment(organizationID)
create_topic_if_needed(organizationID)
subscribe_if_needed(organizationID)
Publish write_assessment(organizationID)
Notify write_assessment(organizationID)
Assessment (SD)
BC_Client
Network_node
Monitor
MonitorClient
AutomaticVerification
Auditor

Figure 15. TWS System Recording sequence diagram

4.2.2.1.2 System Verification

Figure 16 shows the sequence diagram of the *TWS Verification* component. Supposing that in a previous step *TWS Recording* has recorded evidence in the Blockchain, an Auditor could want to check their integrity. For that, it uses the User Interface component, *EmeraldUI,* that calls the *TWS Verification* API. When required, the *TWS Verification* requests the current values of evidence stored in the *Assessment* component - the EMERALD's internal evidence storage-, calculates the hash and compares it with the hash of the same evidence previously recorded in the Blockchain. The validation result can be true or false.

The same process that happens for the evidence can be replicated for the assessment results.

In the case of the automatic verification, it is not the Auditor user, through *EmeraldUI,* who calls the required components, retrieves hashes and makes the manual checking. In this case it only calls the *TWS Verification,* which includes a sub-component that executes the required process to retrieve the actual evidence -from the *Assessment* -, calculate its hash, and compare it with the stored evidence hash.

The same automatic check process is replicated for the assessment results.

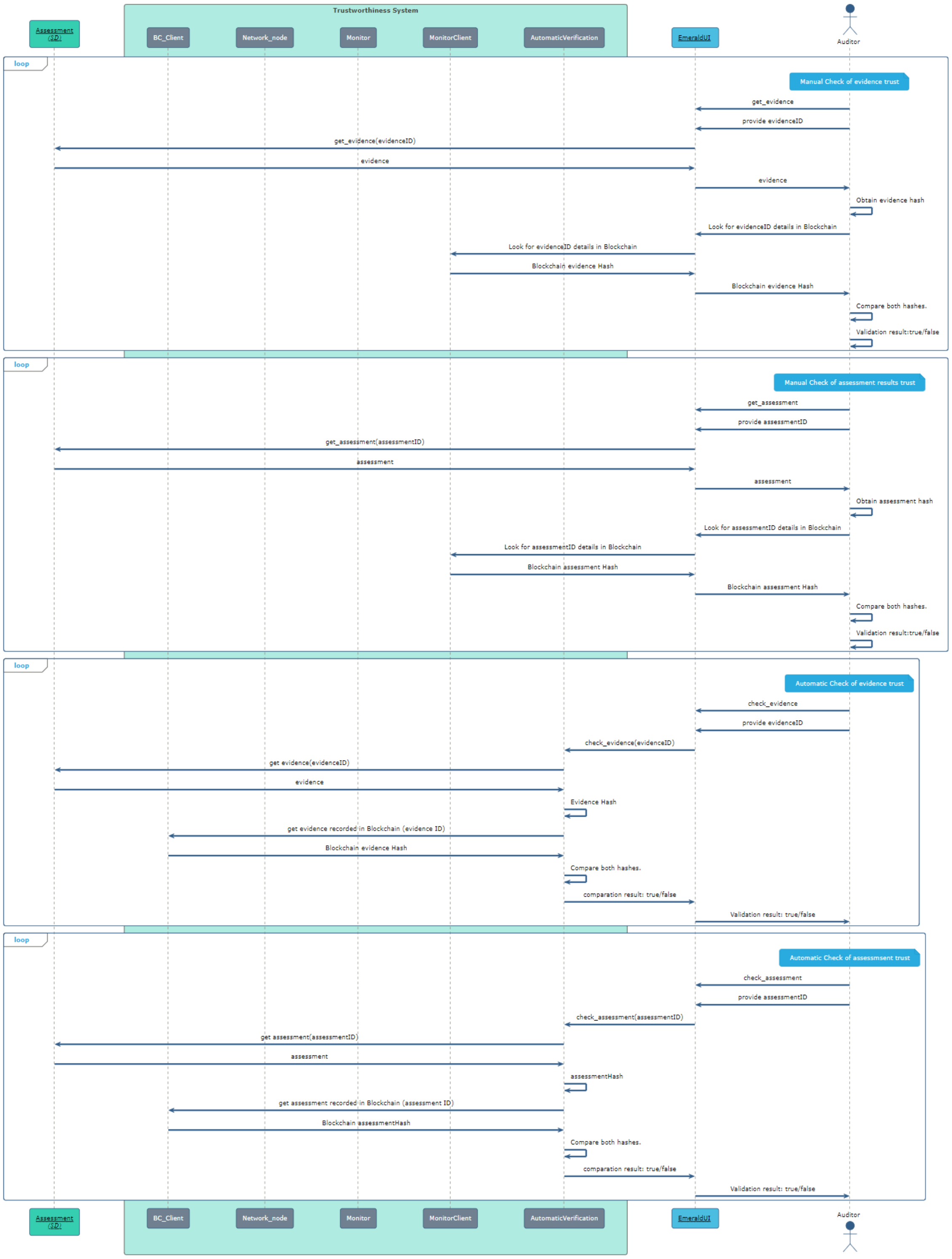

Trustworthiness System
Assessment (SD)
BC_Client
Network_node
Monitor
MonitorClient
AutomaticVerification
EmeraldUI
Auditor
loop
Manual Check of evidence trust
get_evidence
provide evidenceID
get_evidence(evidenceID)
evidence
evidence
Obtain evidence hash
Look for evidenceID details in Blockchain
Look for evidenceID details in Blockchain
Blockchain evidence Hash
Blockchain evidence Hash
Compare both hashes.
Validation result:true/false
loop
Manual Check of assessment results trust
get_assessment
provide assessmentID
get_assessment(assessmentID)
assessment
assessment
Obtain assessment hash
Look for assessmentID details in Blockchain
Look for assessmentID details in Blockchain
Blockchain assessment Hash
Blockchain assessment Hash
Compare both hashes.
Validation result:true/false
loop
Automatic Check of evidence trust
check_evidence
provide evidenceID
check_evidence(evidenceID)
get evidence(evidenceID)
evidence
Evidence Hash
get evidence recorded in Blockchain (evidence ID)
Blockchain evidence Hash
Compare both hashes.
comparation result: true/false
Validation result: true/false
loop
Automatic Check of assessmsent trust
check_assessment
provide assessmentID
check_assessment(assessmentID)
get assessment(assessmentID)
assessment
assessmentHash
get assessment recorded in Blockchain (assessment ID)
Blockchain assessmentHash
Compare both hashes.
comparation result: true/false
Validation result: true/false
Assessment (SD)
BC_Client
Network_node
Monitor
MonitorClient
AutomaticVerification
EmeraldUI
Auditor

Figure 16. TWS System Verification sequence diagram

4.2.3 MARI - Mapping Assistant for Regulations with Intelligence

Component Name	Mapping Assistant for Regulations with Intelligence (MARI)
Main functionalities	The component creates an automatic association between: ● A security control and a security metric. ● Two security controls from two different certification schemes.

<table>
<tr><td><b>Sub-components Description</b></td><td><ul><li><b>Feature extractor</b>, based on a state-of-the-art NLP pre-trained model for transforming textual descriptions of metrics and controls into feature vectors.</li><li><b>Clustering tool</b>, for obtaining metric-control associations.</li></ul></td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><table><tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr><tr><td>API</td><td>API to access MARI functionalities</td><td>REST API</td></tr></table></td></tr>
<tr><td><b>Interaction with other components</b></td><td><ul><li><i>Repository of Controls and Metrics (RCM):</i> MARI reads controls and metrics from the <i>RCM</i> and produces associations, which are then stored back in the <i>RCM</i>.</li><li><i>EMERALD UI:</i> MARI will interface with the <i>EMERALD UI</i> developed in WP4, through which it will be possible to view the results of control/metric associations and control/control associations.</li></ul></td></tr>
<tr><td><b>Relevant sequence diagram/s</b></td><td>See section 4.2.3.1</td></tr>
<tr><td><b>Requirements Mapping</b></td><td><ul><li>MARI.01: AI-based</li><li>MARI.02: Automatic association</li><li>MARI.03: Performance Evaluation</li><li>MARI.04: Usage and Visualization</li><li>MARI.05: Strategies</li></ul></td></tr>
<tr><td><b>Technology used</b></td><td>Python</td></tr>
<tr><td><b>Related KR</b></td><td>KR3_OPTIMA</td></tr>
<tr><td><b>WP and task</b></td><td>WP3 – T3.3</td></tr>
<tr><td><b>License</b></td><td>Open Source with license Apache 2.0</td></tr>
<tr><td><b>Partner</b></td><td>CNR</td></tr>
</table>

4.2.3.1 Sequence diagram

Figure 17 shows the sequence diagram of the *MARI* component. *MARI* is an intelligent system capable of selecting the optimal set of metrics to evaluate the cloud system's compliance within the certification schemes.

The Compliance Manager triggers *MARI*, that will call the *RCM* to obtain the controls and metrics stored there. After the analysis, MARI will return the control/control associations and the control/metric associations to the *RCM*.

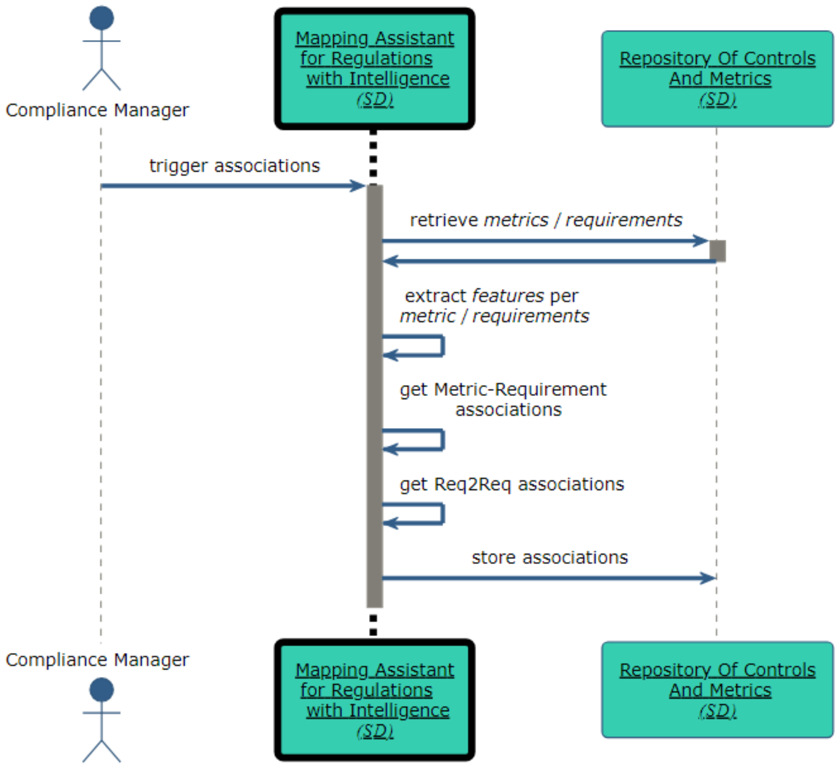

Compliance Manager
Mapping Assistant for Regulations with Intelligence (SD)
Repository Of Controls And Metrics (SD)
trigger associations
retrieve metrics / requirements
extract features per metric / requirements
get Metric-Requirement associations
get Req2Req associations
store associations
Compliance Manager
Mapping Assistant for Regulations with Intelligence (SD)
Repository Of Controls And Metrics (SD)

Figure 17. MARI sequence diagram

4.2.4 RCM - Repository of Controls and Metrics

Component Name	Repository of Controls and Metrics (RCM)
Main functionalities	The component provides the following functionalities: • Stores and manages certification schemes, supporting multi-scheme and multi-level certification. The RCM also incorporates the definition of the metrics used in EMERALD to assess evidence. • The RCM provides mechanisms to update the catalogues and maintain a versioning system and will allow importing and exporting catalogues into/from the RCM using OSCAL as exchange format. • Manages other related information, such as the controls mappings provided by the MARI component, the control implementation guidelines and a self-assessment questionnaire to assess compliance with a scheme.
Sub-components Description	**Frontend**: This sub-component contains the graphical user interface of the RCM (It will be part of the *EmeraldUI* component and communicate with the backend via the API). It allows users to filter the view and select the set of information they want to check from the existing schemes (e.g., controls of a certain scheme, requirements of a certain assurance level, metrics related to some controls, etc). **Backend**: is the core sub-component of the RCM. It implements the APIs to perform the actual management of the scheme data, considering the filters set by the user through the UI or by calling the API. The RCM will contain two backends: i) *Backend converter*, which is dedicated to the scheme conversions to/from OSCAL, and ii) *Backend*, which deals with the management of schemes and metrics.

<table>
<tr><td></td><td><b>Registry</b>: sub-component provided by the framework. Ties the other sub-components together and enables them to communicate with each other.</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>Schema</td><td>Retrieves information about a certification scheme (metrics, requirements, controls, etc) as needed</td><td>Rest API</td></tr>
<tr><td>Mapping</td><td>Sets a control mapping among schemes, provided by the MARI component</td><td>Rest API</td></tr>
<tr><td>Import-export</td><td>Manages import/export of schemes in OSCAL</td><td>Rest API</td></tr>
</table></td></tr>
<tr><td><b>Interaction with other components</b></td><td>• <i>Clouditor-Orchestrator</i><b>,</b> which retrieves the information about the schemes and the metrics from the <i>RCM</i>.
• <i>Mapping Assistant for Regulations with Intelligence (MARI)</i>, which provides the results of the mapping functionality to the <i>RCM</i> in order to store the results for further uses.
• <i>EmeraldUI,</i> which retrieves the information from the <i>RCM</i> to present it in the User Interface. On the other hand, the user may want to introduce new schemes, new versions of a scheme, or answer the self-assessment questionnaire.
• <i>AMOE</i>, a knowledge extractor that obtains from the <i>RCM</i> the definition of the security metrics needed to evaluate evidence from policy documents.</td></tr>
<tr><td><b>Relevant sequence diagram/s</b></td><td>See Section 4.2.4.1</td></tr>
<tr><td><b>Requirements Mapping</b></td><td>The requirements covered by this component are:
• RCM.01: Multi-schema support
• RCM.02: Accessible by the rest of components
• RCM.03: Include metrics for all schemes supported
• RCM.04: Mapping of schemes
• RCM.05: Import/export of security schemes in OSCAL
• RCM.06: Import/export of security schemes in CSV format
• RCM.07: Support for personalized catalogues
• RCM.08: Support updating/versioning of schemes</td></tr>
<tr><td><b>Technology used</b></td><td>Microservices architecture bassed in a jHipster framework:
• Backend side with Java stack with Spring Boot
• Frontend with Angular and Bootstrap</td></tr>
<tr><td><b>Related KR</b></td><td>KR7: INTEROP</td></tr>
<tr><td><b>WP and task</b></td><td>WP3 – T3.2</td></tr>
<tr><td><b>License</b></td><td>Apache license v2.0</td></tr>
<tr><td><b>Partner</b></td><td>TECNALIA</td></tr>
</table>

4.2.4.1 Sequence diagram

Figure 18 shows the sequence diagram of the *RCM* component. The *RCM* provides a central point in the EMERALD framework where the certification schemes are stored and managed. It also incorporates the definition of the metrics used in EMERALD.

In the configuration, the *RCM* can receive partial updates or totally new schemes from the Compliance Manager, who can also consult the schemes in the provided User Interface.

During the runtime, RCM can receive calls from *Orchestrator* and *AMOE***,** which retrieve the information about the schemes and the metrics from the *RCM*. The *RCM* can also receive a call from *MARI* which, in turn, has been called by the user to calculate a mapping of controls or a mapping among controls and metrics. In this case, *MARI* retrieves the input information from the *RCM*, and after processing it, returns the map to be stored in the *RCM*.

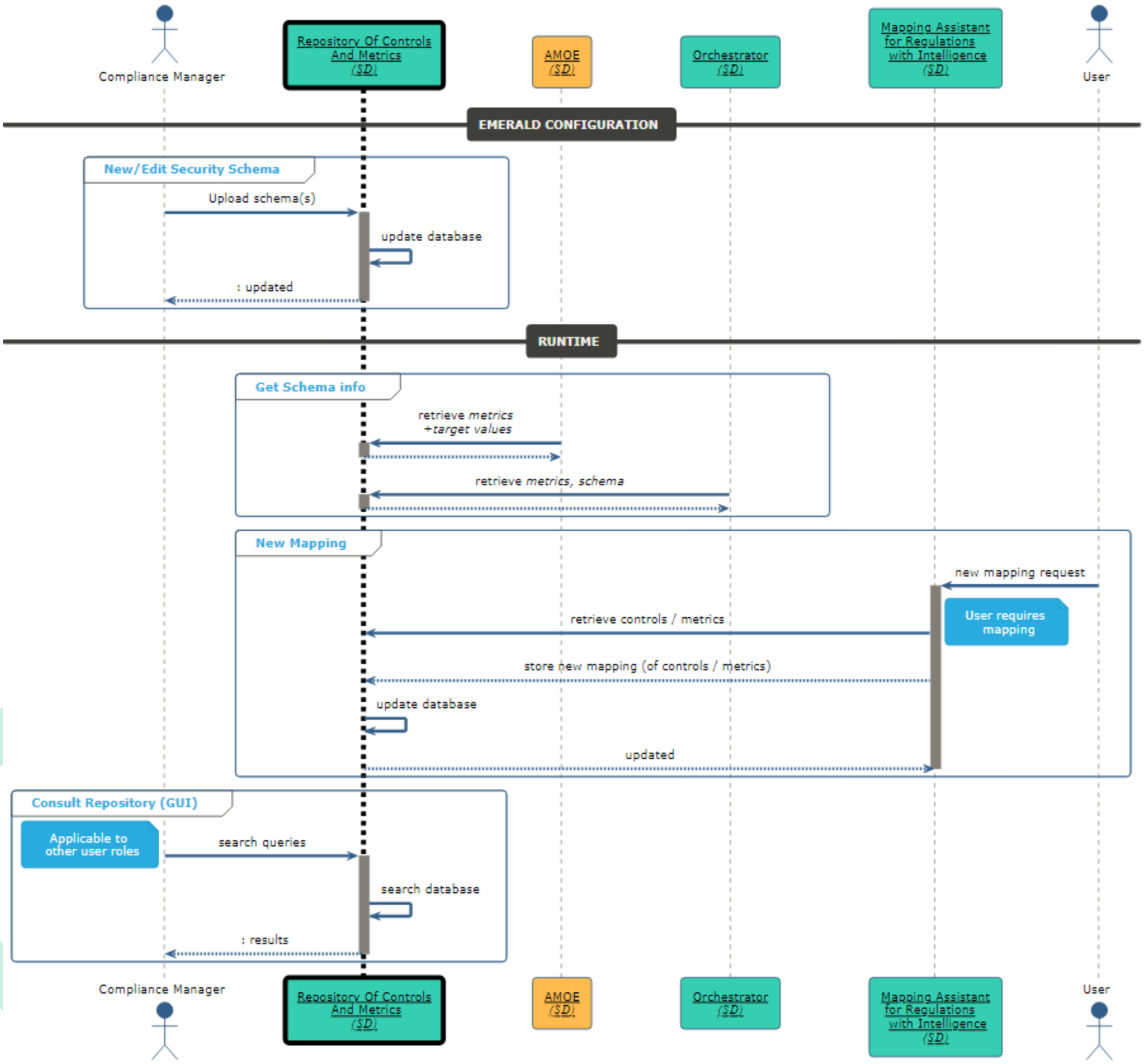


Figure 18. RCM sequence diagram

4.2.5 Orchestrator

Component Name	Orchestrator
Main functionalities	The component provides the following functionalities:

<table>
<tr><td></td><td><ul><li>Orchestrator is the central component for connecting multiple components together and to receive information, e.g., assessment results.</li><li>Makes final certification decisions.</li></ul></td></tr>
<tr><td><b>Sub-components Description</b></td><td><ul><li><i>Currently no division in subcomponents planned</i></li></ul></td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td>All APIs are available via REST and gRPC.
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>CLI</td><td>A CLI is available</td><td>Cobra[18]/Viper[19]</td></tr>
<tr><td>gRPC API</td><td>The following endpoints are available:<ul><li>ListAssessmentResults lists stored assessment results.</li><li>StoreAssessmentResult stores a given assessment result.</li><li>StoreAssessmentResults stores a stream of assessment results.</li><li>GetMetric returns the metric for the given metric ID</li><li>ListMetrics lists all metrics provided by the given metric catalogue</li></ul></td><td>gRPC</td></tr>
</table></td></tr>
<tr><td><b>Interaction with other components</b></td><td><ul><li><i>EmeraldUI</i>: The <i>EmeraldUI</i> retrieves relevant information from the Orchestrator (e.g., assessment results, certification decisions, certification schemes).</li><li><i>Evaluation</i><ul><li>The <i>Evaluation</i> component registers with the <i>Orchestrator</i> (not yet implemented, to be discussed).</li><li>The <i>Evaluation</i> component retrieves assessment results from the <i>Orchestrator</i> for the evaluation.</li></ul></li><li><i>Assessment</i><ul><li>The <i>Assessment</i> component registers with the <i>Orchestrator</i> (not yet implemented, to be discussed).</li><li>The <i>Assessment</i> sends the assessment results to the <i>Orchestrator</i> for storage.</li></ul></li><li><i>Repository of Controls and Metrics</i>: The <i>Orchestrator</i> retrieves metrics and controls.</li><li><i>Evidence Store</i><ul><li>The <i>Evidence Store</i> component registers with the <i>Orchestrator</i> (not yet implemented, to be discussed).</li></ul></li></ul></td></tr>
</table>

[18] https://github.com/spf13/cobra

[19] https://github.com/spf13/viper

Relevant sequence diagram/s	See Section 4.2.5.1
Requirements Mapping	List of requirements covered by this component: • ORCH.01: Final certificate decision • ORCH.02: REST API Gateway for UI • ORCH.03: Role Based Access Control • ORCH.04: Manage Tools (such as Evidence Extractors) via API • ORCH.05: Provide an API for audit workflow
Technology used	Go[20], gRPC[21]
Related KR	KR4_MULTICERT KR6_EMERALD UI/UX
WP and task	WP3 – T3.1
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.5.1 Sequence diagram

Figure 19 shows the sequence diagram of the *Orchestrator* component. The *Orchestrator* is the central component orchestrating the certification process and connecting multiple components together of the EMERALD framework.

The *Orchestrator* accesses the *RCM* to retrieve relevant metrics and controls (as well as the respective mapping provided by *MARI*).

The *Orchestrator* receives assessment results sent by the *Assessment* component which are then stored in the internal database. In the same sense, the *Evaluation* component sends the evaluation results to the *Orchestrator*, to get them stored in the internal database.

When asked from the *EmeraldUI* or any other component, the *Orchestrator* can also fetch the stored data from the internal database and return it.

[20] https://go.dev/
[21] https://grpc.io/

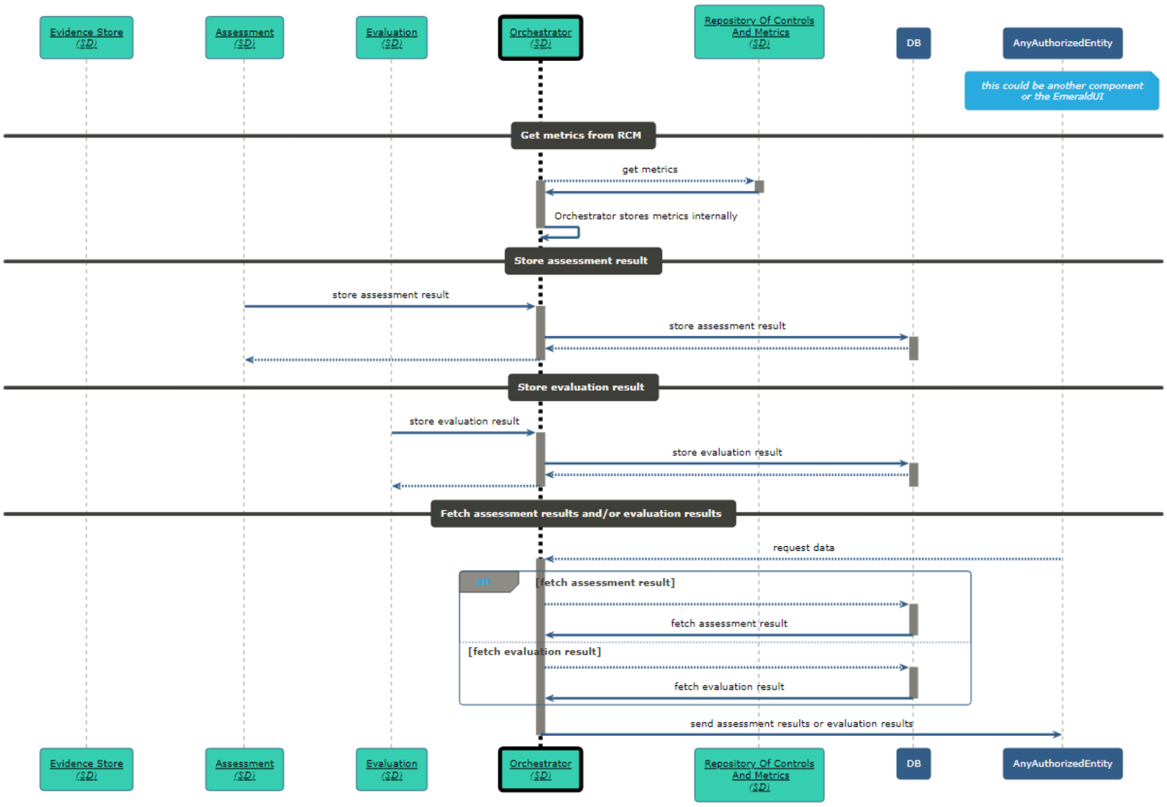


Figure 19. Orchestrator sequence diagram

4.2.6 Evidence Store

<table>
<tr><td><b>Component Name</b></td><td colspan="3">Evidence Store</td></tr>
<tr><td><b>Main functionalities</b></td><td colspan="3">The component provides the following functionalities:
• Stores and manages evidence of different resource types received from various discovery components as well as assessment results.</td></tr>
<tr><td><b>Sub-components Description</b></td><td colspan="3">Currently no division in subcomponents planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td><b>Interface name</b></td><td><b>Description</b></td><td><b>Interface technology</b></td></tr>
<tr><td></td><td>CLI</td><td>A CLI is available</td><td>Cobra[22]/Viper[23]</td></tr>
<tr><td></td><td>REST API/gRPC API</td><td>The following endpoints are available:
• GetEvidence for receiving specific evidence
• ListEvidences to list multiple evidence
• StoreEvidence to store one evidence
• StoreEvidences to store multiple evidence in a stream</td><td>All endpoints are available via the REST API and gRPC API</td></tr>
</table>

[22] https://github.com/spf13/cobra

[23] https://github.com/spf13/viper

Interaction with other components	• *Assessment*: Forwards the evidence to the *Assessment* • *AI-SEC*: Evidence Store receives evidence from *AI-SEC* • *Codyze*: Evidence Store receives evidence from *Codyze* • *Clouditor-Discovery*: Evidence Store receives evidence from *Clouditor-Discovery* • *eknows*: Evidence Store receives evidence from *eknows* • *AMOE*: Evidence Store receives evidence from *AMOE* • *Orchestrator* ◦ Fetches evidence from *Evidence Store* ◦ Registers *Evidence Store* component in the *Orchestrator* (not yet implemented, to be discussed).
Relevant sequence diagram/s	See section 4.2.6.1
Requirements Mapping	List of requirements covered by this component: • ESTORE.01: Storage of evidence as ontology entities in graph database • ESTORE.02: Allow Interaction with Third-Party Tools
Technology used	Go[24], gRPC (using protobuf)[25], a specific database to implement the knowledge graph (tbd)
Related KR	KR1_EXTRACT KR2_CERTGRAPH
WP and task	WP3 – T3.1
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.6.1 Sequence diagram

Figure 20 shows the sequence diagram of the *Evidence Store* component. The *Evidence Store* component is responsible for storing and managing evidence of different resource types and collected from various sources in a graph database.

First thing the *Evidence Store* does is to register itself in the *Orchestrator,* so retrieves meta data (e.g., the cloud services identification) to add this data to the incoming evidence.

Various evidence collectors (like the *Clouditor-Discovery* in the diagram) gather evidence from different sources and send them to the *Evidence Store*.

When required, the *Assessment* component pulls the required evidence from the *Evidence Store* for its assessment calculation.

[24] https://go.dev/

[25] https://grpc.io/

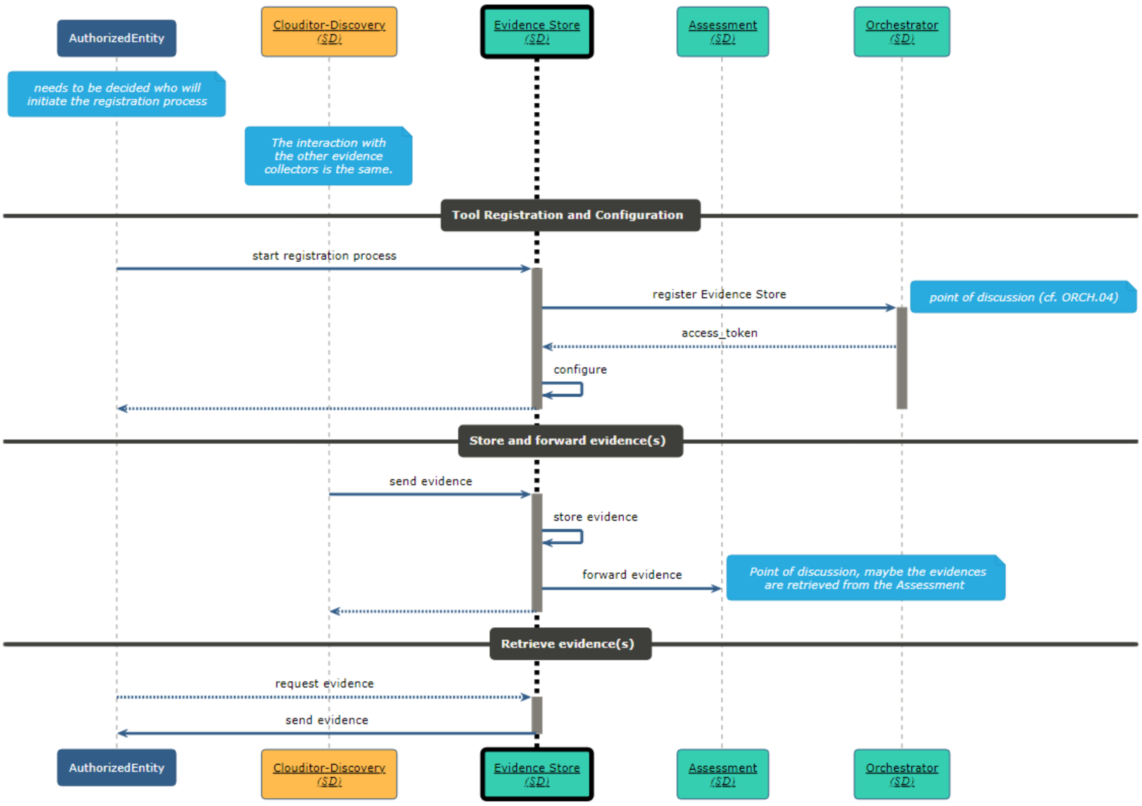

AuthorizedEntity
Clouditor-Discovery (SD)
Evidence Store (SD)
Assessment (SD)
Orchestrator (SD)
needs to be decided who will initiate the registration process
The interaction with the other evidence collectors is the same.
Tool Registration and Configuration
start registration process
register Evidence Store
point of discussion (cf. ORCH.04)
access_token
configure
Store and forward evidence(s)
send evidence
store evidence
forward evidence
Point of discussion, maybe the evidences are retrieved from the Assessment
Retrieve evidence(s)
request evidence
send evidence
AuthorizedEntity
Clouditor-Discovery (SD)
Evidence Store (SD)
Assessment (SD)
Orchestrator (SD)

Figure 20. Evidence Store sequence diagram

4.2.7 Assessment

<table>
<tr><td><b>Component Name</b></td><td>Assessment</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:
• Assesses evidence based on predefined metrics that are stored in the <i>Repository of Controls and Metrics.</i></td></tr>
<tr><td><b>Sub-components Description</b></td><td>Currently no division in subcomponents planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td>
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>CLI</td><td>A CLI is available</td><td>Cobra[26]/Viper[27]</td></tr>
<tr><td>REST API/ gRPC API</td><td>The following endpoints are available:
• AssessEvidence to assess one evidence.
• AssessEvidences to assess a stream of evidence.</td><td>All endpoints are available via the REST API and gRPC API.</td></tr>
</table>
</td></tr>
</table>

[26] https://github.com/spf13/cobra

[27] https://github.com/spf13/viper

Interaction with other components	• *Evidence Store*: The *Assessment* retrieves evidence from the *Evidence Store*. • *Orchestrator*: • Registers the *Assessment* component in the *Orchestrator* (not yet implemented, to be discussed). • The *Assessment* sends the assessment results to the *Orchestrator* for storage. • The *Assessment* retrieves the metrics for the assessment from the *Orchestrator*. • *Trustworthiness System*: The Assessment component sends evidence and assessment results to the *Trustworthiness System*.
Relevant sequence diagram/s	See Section 4.2.7.1
Requirements Mapping	List of requirements covered by this component: • ASSESS.01: Assessment based on evidence • ASSESS.02: Assessment rules for 80% of the defined metrics • ASSESS.03: Display cause of assessment result
Technology used	Go[28], gRPC (using protobuf)[29], Rego (Open Policy Agent)[30]
Related KR	KR4_MULTICERT KR6_EMERALD UI/UX
WP and task	WP3 – T3.4
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.7.1 Sequence diagram

Figure 21 shows the sequence diagram of the *Assessment* component. The *Assessment* component is responsible for assessing evidence based on predefined metrics. The calculated assessment results are eventually used by the *Clouditor-Evaluation* component to determine compliance with the relevant controls.

At an initial registration phase, the *Assessment* component coordinates with the *Orchestrator* to receive instructions.

The *Assessment* retrieves evidence from the *Evidence Store* to perform assessments. The result of the assessment is sent to the *Orchestrator* for storage.

The *Assessment* interacts with the *TWS* to provide assessment results as well as the respective evidence.

[28] https://go.dev/

[29] https://grpc.io/

[30] https://www.openpolicyagent.org/docs/latest/policy-language/

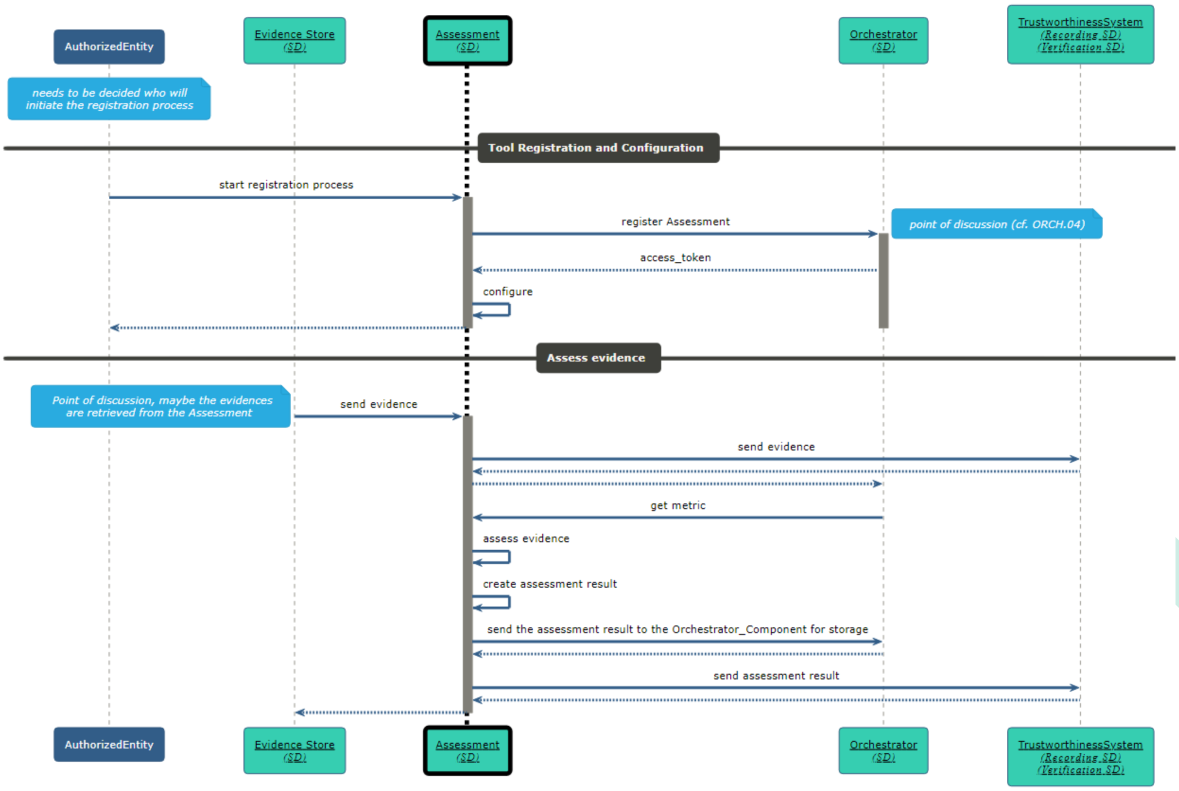

AuthorizedEntity
Evidence Store (SD)
Assessment (SD)
Orchestrator (SD)
TrustworthinessSystem (Recording SD) (Verification SD)
needs to be decided who will initiate the registration process
Tool Registration and Configuration
start registration process
register Assessment
point of discussion (cf. ORCH.04)
access_token
configure
Assess evidence
Point of discussion, maybe the evidences are retrieved from the Assessment
send evidence
send evidence
get metric
assess evidence
create assessment result
send the assessment result to the Orchestrator_Component for storage
send assessment result
AuthorizedEntity
Evidence Store (SD)
Assessment (SD)
Orchestrator (SD)
TrustworthinessSystem (Recording SD) (Verification SD)

Figure 21. Assessment sequence diagram

4.2.8 Evaluation

<table>
<tr><td><b>Component Name</b></td><td>Evaluation</td></tr>
<tr><td><b>Main functionalities</b></td><td>The component provides the following functionalities:
• Aggregates assessment results assed by the Assessment component and determines the overall compliance status for a given control.
• Evaluates the compliance of cloud services against controls and requirements of security catalogues.</td></tr>
<tr><td><b>Sub-components Description</b></td><td>Currently no division in subcomponents planned</td></tr>
<tr><td><b>Main logical Interfaces offered</b></td><td>
<table>
<tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr>
<tr><td>CLI</td><td>A CLI is available</td><td>Cobra[31]/Viper[32]</td></tr>
<tr><td>REST API/gRPC API</td><td>The following endpoints are available:
• StartEvaluation starts the evaluation.
• ListEvaluationResults lists stored evaluation results.</td><td>All endpoints are available via the REST API and gRPC API.</td></tr>
</table>
</td></tr>
</table>

[31] https://github.com/spf13/cobra

[32] https://github.com/spf13/viper

Interaction with other components	• *Orchestrator* • Registers the *Evaluation* component in the *Orchestrator* (not yet implemented). • The *Evaluation* component retrieves assessment results from the *Orchestrator*. • Sends the evaluation results to the *Orchestrator* for storage. • Fetches controls from the *Orchestrator*.
Relevant sequence diagram/s	See Section 4.2.8.1
Requirements Mapping	List of requirements covered by this component: • EVAL.01: Display cause of evaluation result • EVAL.02: Evaluation based on assessment results
Technology used	Go[33], gRPC[34]
Related KR	KR4_MULTICERT KR6_EMERALD UI/UX
WP and task	WP3 – T3.4
License	Apache-2.0
Partner	Fraunhofer AISEC

4.2.8.1 Sequence diagram

Figure 22 shows the sequence diagram of the *Evaluation* component. The *Evaluation* component is responsible for aggregating and interpreting assessment results to determine overall compliance status of cloud services for a given control of a security catalogue.

The *Evaluation* first registers itself into the *Orchestrator.*

The *Evaluation* component obtains assessment results from the *Orchestrator,* processes them and determines the compliance status based on the mapping of metrics to controls of a security catalogue. The evaluation result is sent back to the *Orchestrator* for storage.

[33] https://go.dev/

[34] https://grpc.io/

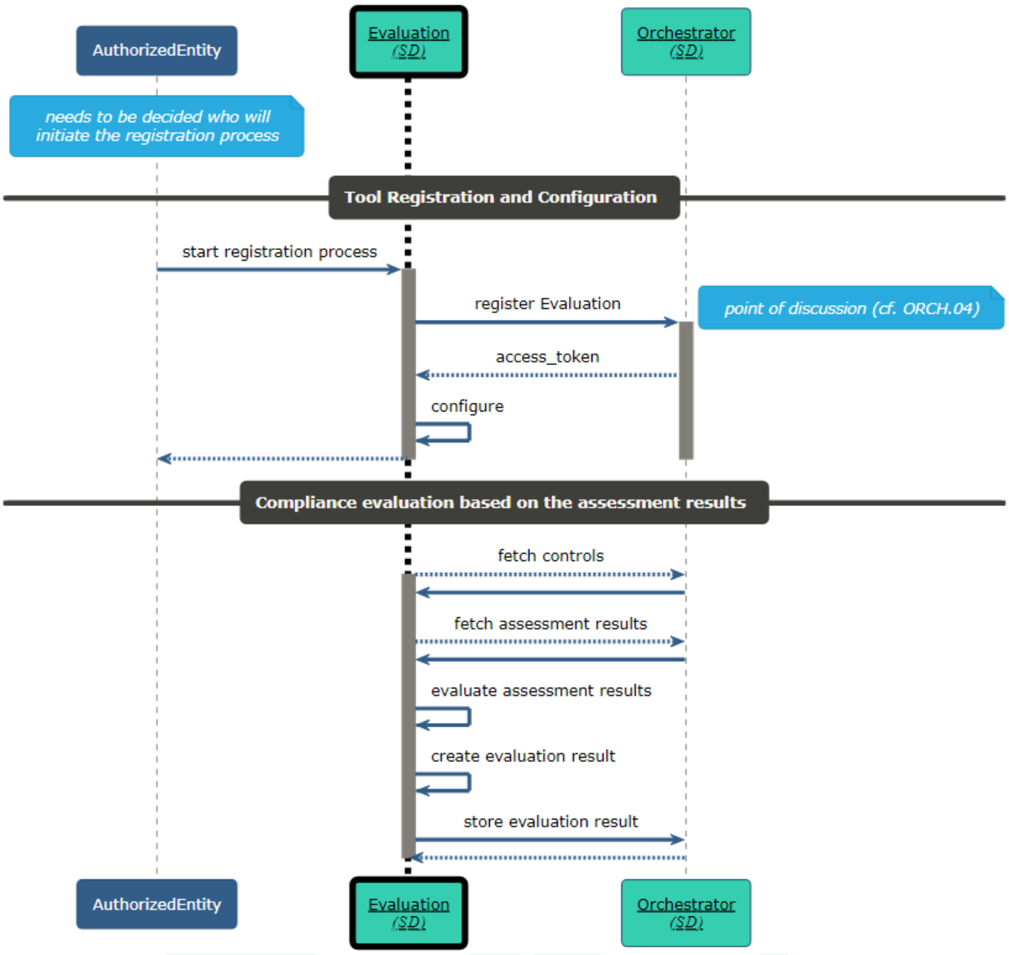

AuthorizedEntity
Evaluation (SD)
Orchestrator (SD)
needs to be decided who will initiate the registration process
Tool Registration and Configuration
start registration process
register Evaluation
point of discussion (cf. ORCH.04)
access_token
configure
Compliance evaluation based on the assessment results
fetch controls
fetch assessment results
evaluate assessment results
create evaluation result
store evaluation result
AuthorizedEntity
Evaluation (SD)
Orchestrator (SD)

Figure 22. Evaluation sequence diagram

5 Conclusions

This document is dedicated to introducing the EMERALD architecture to the reader. An overview of the system, the decomposition of EMERALD in 12 components, the information flow among them and a detailed view of them have been provided. These components will be in the future instantiated in the pilots defined in WP5. To complement the architecture, the general data model of the EMERALD framework, defined in D1.1 [1], has been presented. A Glossary is also included, with definition and examples of crucial terms.

Following a multiple-perspective process, the requirements for the EMERALD framework have been designed. This document focuses on technical requirements, but we also included the Business requirement list, developed in WP5, and the UX/UI requirements, developed in WP4, for completion and analysis. A total of 44 functional requirements have been elicited, grouped in the 12 components that form the framework.

These functional requirements are accompanied by 8 non-functional requirements, which are mostly system constrains or properties more than related to a particular component, so no effort has been spent in linking them to specific components. For each NFR, some hints on how we plan to fulfil them have been presented.

An analysis of the requirements has been provided, where several matrices trace the coverage provided by the requirements to validate the pilots, the Key Results (KRs) or the Key Performance Indicators (KPIs). Also, the requirements prioritization and status at this V1 version of the EMERALD components in M12 is analysed. As a result, we have demonstrated that most of the Business requirements are covered by one or more technical requirements. That means that the corresponding component design is aligned with the final user's view. Finally, we have provided a detailed view of the EMERALD framework, describing each component based on the component cards, which included sequence diagram developed with PlantUML to show their dynamic behaviour and interaction with other components.

The future version of this document (D1.4 [2]) will review these requirements, their status and mappings, and could include new requirements as a result of the evolution of components, or of task related to the technical and pilots' validation activities.

6 References

[1] EMERALD Consortium, “D1.1 Data modelling and interaction mechanisms-v1,” 2024.

[2] EMERALD Consortium, “D1.4 EMERALD solution architecture-v2,” 2025.

[3] EMERALD Consortium, “D1.7 EMERALD Integrated solution–v1,” 2025.

[4] EMERALD Consortium, “D4.1 Results of the UI-UX requirements analysis and the work processes–v1,” 2024.

[5] EMERALD Consortium, “D4.3 - User interaction and user experience,” 2024.

[6] European Comission, “Regulation (EU) 2019/881 of the European Parliament and of the Council of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No 52,” 10 2024. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2019/881/oj. [Accessed 10 2024].

[7] ISO, “ISO 9000:2015(en), Quality management systems — Fundamentals and vocabulary,” https://www.iso.org/obp/ui#iso:std:iso:9000:ed-4:v1:en, 2015.

[8] “ISO/IEC 17788:2014 - Information technology — Cloud computing — Overview and vocabulary,” 2014.

[9] National Institute of Standards and Technology (NIST), “SECURITY AND PRIVACY CONTROLS FOR INFORMATION SYSTEMS AND ORGANIZATIONS,” 2020.

[10] ISO, “ISO/IEC 27000:2018 Information technology — Security techniques — Information security management systems — Overview and vocabulary,” 2018.

[11] NIST - National Institute of Standards and Technology, “Key Concepts and Terms Used in OSCAL,” 10 2024. [Online]. Available: https://pages.nist.gov/OSCAL/resources/concepts/terminology/. [Accessed 10 2024].

[12] NIST - National Institute of Standards and Technology, “Cloud Computing Service Metrics Description,” 24 April 2018. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-307.pdf. [Accessed 10 2024].

[13] EMERALD Consortium, “D2.2 - Source Evidence Extractor – v1,” 2024.

[14] EMERALD Consortium, “D2.4 - AMOE – v1,” 2024.

[15] EMERALD Consortium, “D2.6 - ML model certification – v1,” 2024.

[16] EMERALD Consortium, “D2.8 Runtime evidence extractor – v1,” 2024.

[17] EMERALD Consortium, “D3.1 Evidence assessment and Certification–Concepts-v1,” 2024.

[18] EMERALD Consortium, “D5.1 Pilot definition, set-up & validation plan,” 2024.

[19] EMERALD Consortium, “EMERALD - Annex 1 - Description of Action - GA 101120688,” 2022.

APPENDIX A: Current status of requirements

Table 14 depicts the status of the technical requirements, ordered by component. The peach-coloured lines highlight those requirements that are foreseen for M12.

The "Timeline" column states the month foreseen to complete the implementation, and the associated Milestone (see the codes below) in an abbreviated form, where "C" stands for the Components version, and "I" stands for Integration version. For example:

- C-v1 = MS2: Components V1 (M12)
- I-v3 = MS8: Integrated audit suite V3 (M34)

Table 14. Status of the Technical requirements

Req. ID	Title	Priority	Timeline	Status
AI-SEC.01	The extractor tool includes selected criteria	MUST	M12 (C-v1)	35%
AMOE.01	Upload PDF document	MUST	M12 (C-v1)	90%
AMOE.02	Provision of extracted evidence to EvidenceStore (Orchestrator/Clouditor)	MUST	M24 (C-V2)	50%
AMOE.03	Refine evidence extraction approach	MUST	M24 (C-V2)	0%
AMOE.04	Compare results from multiple documents	SHOULD	M12 (C-v1)	70%
AMOE.05	Select metrics per document	SHOULD	M24 (C-V2)	0%
AMOE.06	Classify document, select respective metrics (optional)	MUST	M34 (I-v3)	0%
AMOE.07	Metric states	SHOULD	M24 (C-V2)	0%
CLDISC.01	Discovery of security properties of infrastructure components	MUST	M30 (I-v2)	40%
CODYZE.01	Extraction of security features from source code	MUST	M30 (I-v2)	20%
EKNOWS.01	Integration into existing systems	MUST	M18 (I-v1)	30%
EKNOWS.02	Resilience while analysing erroneous code	SHOULD	M24 (C-V2)	70%
EKNOWS.03	Multi-language support	MUST	M24 (C-V2)	50%
EKNOWS.04	Support EMERALD evidence format	MUST	M18 (I-v1)	0%
EKNOWS.05	Static code analysis	MUST	M24 (C-V2)	60%
TWS.01	Provide integrity proof of evidence	MUST	M12 (C-v1)	75%
TWS.02	Provide integrity proof of assessment results	MUST	M12 (C-v1)	75%
TWS.03	Provide access through REST API or graphical interface	MUST	M24 (C-V2)	50%
TWS.04	Use a general purpose public-private Blockchain network	MUST	M24 (C-V2)	5%
MARI 1.0	AI-based	MUST	M30 (I-v2)	15%
MARI 2.0	Automatic association	MUST	M30 (I-v2)	15%
MARI 3.0	Performance evaluation	MUST	M30 (I-v2)	15%
MARI 4.0	Usage and visualization	MUST	M30 (I-v2)	15%

Req. ID	Title	Priority	Timeline	Status
MARI 5.0	Strategies	MUST	M30 (I-v2)	15%
RCM.01	Multi-schema support	MUST	M12 (C-v1)	90%
RCM.02	Accessible by the rest of components	MUST	M12 (C-v1)	100%
RCM.03	Include metrics for all schemas supported	MUST	M12 (C-v1)	30%
RCM04	Mapping of schemes	SHOULD	M30 (I-v2)	10%
RCM.05	Import/export of security schemes in OSCAL	MUST	M30 (I-v2)	40%
RCM.06	Import/export of security schemes in CSV format	COULD	M12 (C-v1)	60%
RCM.07	Support for personalized catalogues	MUST	M30 (I-v2)	0%
RCM.08	Support updating/versioning of schemes	SHOULD	M30 (I-v2)	10%
ORCH.01	Final certificate decision	MUST	M24 (C-v2)	0%
ORCH.02	REST API Gateway for UI	MUST	M12 (C-v1)	15%
ORCH.03	Role Based Access Control	MUST	M24 (C-v2)	25%
ORCH.04	Manage Tools (such as Evidence Extractors) via API	MUST	M18 (I-v1)	0%
ORCH.05	IssueORCH.05 Provide an API for audit workflow	MUST	M30 (I-v2)	0%
ESTORE.01	Storage of ontology entities in graph database	MUST	M18 (I-v1)	15%
ESTORE.02	Allow Interaction with Third-Party Evidence Collectors	SHOULD	M34 (I-v3)	15%
ASSESS.01	Assessment based on evidence	MUST	M30 (I-v2)	15%
ASSESS.02	Assessment rules for 80% of the defined metrics	MUST	M30 (I-v2)	15%
ASSESS.03	Display cause of assessment result	COULD	M30 (I-v2)	0%
EVAL.01	Display cause of failing evaluation result	COULD	M30 (I-v2)	0%
EVAL.02	Evaluation based on assessment results	MUST	M30 (I-v2)	15%

The list of Milestones of the EMERALD project are [19]:

- MS1: Project baselines and definition (M9)
- MS2: Components V1 (M12)
- MS3: Integrated audit suite V1 (M18)
- MS4: Pilots V1 (M20)
- MS5: Components V2 (M24)
- MS6: Integrated audit suite V2 (M30)
- MS7: Pilots V2 (M32)
- MS8: Integrated audit suite V3 (M34)
- MS9: Final evaluation report and impact analysis (M36)