ANASTACIA

Advanced Networked Agents for Security and Trust Assessment in CPS/IoT Architectures

# D6.1

## Initial Technical Integration and Validation Report

This deliverable provides the output of Task 6.1. It provides information about the integrated analysis of all software resources available and thereafter defines necessary interfaces to integrate components. An integration plan of the discrete framework's mechanisms and software components is also specified and documented. In addition to the integration plan, a testing and technical evaluation plan is provided.

| | |
|---|---|
| **Distribution level** | PU |
| **Contractual date** | <28.02.2018> [M14] |
| **Delivery date** | <23.03.2018> [M15] |
| **WP / Task** | WP6 / T6.1 |
| **WP Leader** | UBITECH |
| **Authors** | Giannis Ledakis (UBITECH), Ruben Trapero (ATOS), Diego Rivera (MONT), Yacine Khettab, Ivan Farris (AALTO), Dallal Belabed (THALES), Cédric Crettaz (MAND), Jorge Bernal, Alejandro Molina (UMU), Rafael Marín Pérez (ODINS), Alie El-Din Mady (UTRC), Enrico Cambiaso (CNR), Ivan Vaccari (CNR) |
| **EC Project Officer** | Carmen Ifrim<br>carmen.ifrim@ec.europa.eu |
| **Project Coordinator** | Softeco Sismat SpA<br>Stefano Bianchi<br>Via De Marini 1, 16149 Genova – Italy<br>+39 0106026368<br>stefano.bianchi@softeco.it |
| **Project website** | www.ANASTACIA-h2020.eu |

ANASTACIA has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement N° 731558 and from the Swiss State Secretariat for Education, Research and Innovation.

# Table of contents

# Index of figures



# Index of tables

# PUBLIC SUMMARY

The document presents the integration and technical testing plan of the ANASTACIA framework, that is the outcome of the task T.6.1. The integration plan has been created using ANASTACIA Deliverable D1.3 as starting point and, by identifying and specifying the necessary integration points between components, and also by constructing the integration approach that will be followed. The supported functionalities that each of the releases of ANASTACIA framework shall provide were identified, allowing the consortium to make proper scheduling of both development and integration actions.

Analysis of existing models and standards has been done for the preparation of the technical validation of ANASTACIA framework. Based on this analysis, the document provides initial definition of metrics and KPIs for the technical validation of ANASTACIA as framework and also at component level. Also, the definition of integration testing as part of ANASTACIA was provided in this document, along with integration test that have been identified so far.

Finally, the suggested development cycle and the tools that can be used to support both the collaborative development and the integration of the discrete mechanisms have been provided. The document presents also the selection of GitLab as the basic tool that will help transforming the development and integration plans to specific actions.

# 1 INTRODUCTION

## 1.1 AIMS OF THE DOCUMENT

In this document we provide the output of the efforts of creating the integration, testing and technical evaluation plan of ANASTACIA framework. For the creation of the integration plan technical partners collaborated in many different occasions, especially for the definition of the necessary integration points between components, but also for a common approach regarding testing and evaluation. The starting point for the integration plan was ANASTACIA Deliverable D1.3 and the work done in task T1.3: Architectural Design, continued with the specification and development of the mechanisms described in WP2, WP3, WP4 and WP5. The goal of creating the integration plan is to support the development by guiding the integration of the discrete mechanisms and software components, with the agreement on identified interfaces, and the usage of selected tools. Using these tools (especially GitLab) the actual integration will be carried in the next months based on two major releases when the overall framework will be tested and evaluated. Finally, in this document the development lifecycle scheme proposed by ANASTACIA is presented and includes source code management, continuous integration, source-code quality control, release management and ticketing.

## 1.2 APPLICABLE AND REFERENCE DOCUMENTS

This document refers to the following documents:

- Grant Agreement N°731558 – Annex I (Part A) – Description of Action
- D1.2 – User-centred Requirement Initial Analysis
- D1.3 - Initial Architectural Design

## 1.3 REVISION HISTORY

| Version | Date | Author | Description |
|---|---|---|---|
| **0.1** | 14.12.2017 | Giannis Ledakis (UBI) | Skeleton of expected contents |
| **0.4** | 15.01.2018 | Giannis Ledakis (UBI) | Draft content in sections 2, 3 and 5 |
| **0.7** | 22.02.2018 | Giannis Ledakis (UBI) | First draft version with sample on section that required input from partners |
| **0.7.4** | 28.02.2018 | ALL | First integrated draft |
| **0.8** | 12.03.2018 | Giannis Ledakis (UBI) | Updated version based on comments received in the first draft |
| **0.8.4** | 16.03.2018 | ALL | Second integrated draft |
| **0.9** | 19.03.2018 | Giannis Ledakis | Version released for ATOS review |
| **0.9.1** | 21.03.2018 | Ruben Trapero (ATOS) | Revised version |

| | | | | |
|---|---|---|---|---|
| **1.0** | 23.03.2018 | Giannis (UBI) | Ledakis | Final version released, ready to be delivered |

## 1.4 ACRONYMS AND DEFINITIONS

| Acronym | Meaning |
|---|---|
| **API** | Application Programming Interface |
| **CI** | Continuous Integration |
| **DSPS** | Dynamic Security and Privacy Seal |
| **DTLS** | Datagram Transport Layer Security |
| **ESB** | Enterprise Service Bus |
| **ETSI** | European Telecommunications Standards Institute |
| **HSPL** | High Security Policy Language |
| **IoT** | Internet of Things |
| **MANO** | Management and Orchestration |
| **MMT** | Montimage Monitoring Tool |
| **MSPL** | Medium Security Policy Language |
| **MTTR** | Mean Time to Recovery |
| **NFV** | Network Function Virtualization |
| **OS** | Operating System |
| **REST** | REpresentational State Transfer |
| **RPC** | Remote Procedure Call |
| **SDN** | Software Defined Networking |
| **SIEM** | Security Information and Event Management |
| **UI** | User Interface |
| **VM** | Virtual Machine |
| **XACML** | eXtensible Access Control Markup Language |

# 2 PLATFORM INTEGRATION OVERVIEW

A modern software system like ANASTACIA is a combination of different subsystems cooperating so that the overall framework is able to deliver the needed functionalities. These subsystems need to be integrated in such a way that they can support common business processes and data sharing across whole framework. An effective integration of an application should provide efficient, secure and reliable data exchange between multiple components. However, software Integration deals not only with computer network discipline but also with other issues like the technological diversity of the components used in ANASTACIA, including different communication protocols or messaging format (such as RESTful, XML-RPC, etc.) or even issues with authentication and authorization. In the following section the different integration methods that have been examined are provided, followed by the key elements of ANASTACIA integration.

## 2.1 SOFTWARE INTEGRATION APPROACHES

As integrations of multi component systems are an important part of software development, several different methods of integration have been suggested. A few of the most important ones are presented by K. Hammer and T. Timmerman in the "Fundamentals of Software Integration" [2] and are shortly provided here:

- "***Vertical Integration***": The process of integrating subsystems according to their functionality by creating functional entities. This method provides a quickly performed integration by involving only the necessary vendors and therefore this method is cheaper in the short term. However, the cost of integration can be significantly higher than the one seen in other methods, since in case of new or enhanced functionality, the only possible way to scale the system is to implement a new entity. It is not possible to reuse subsystems in order to have new or enhance existing functionalities.
- "***Star Integration***": The process of integrating subsystems, where each system is interconnected to each of the remaining subsystems. By observing from the perspective of the subsystem, which is being integrated, the connections are like a star. The cost of using this method varies due to the interfaces which the subsystems are exporting. If the subsystems are exporting heterogeneous or proprietary interfaces, the integration costs can rise significantly. The needed integration time and costs of the systems increase exponentially when adding new subsystems. This method often seems preferable, due to the extreme flexibility of the reusability.
- "***Horizontal Integration***": The process of integrating subsystems, where a subsystem is exclusively responsible for the communication between other subsystems. Using this method, the number of interfaces is only one per subsystem. This interface connects directly to this type of integration. It is capable of translating the interface into another interface. This cuts the costs of integration and provides extreme flexibility. In addition, it is possible to entirely replace one subsystem with another one, which provides similar functionality but exports different interfaces, all this completely transparent for the rest of the subsystems. If the cost of intermediate data transformation or the cost of shifting responsibility over business logic is thought to be avoided, then the horizontal scheme can be misleading.
- "***Enterprise Application Integration***": The process of integrating systems that usually stipulate an application-independent (or common) data format. This method usually provides a data transformation service in order to convert between application-specific and common formats. This is done in two steps: first, the adapter converts information from the application's format to the common format and second, semantic transformations are applied on this (converting zip codes to city names, splitting/merging objects from one application into objects in the other applications, and so on).

Of course, many other approaches exist and combinations of the different approaches are commonly used depending on the actual needs and facts of the integrated systems. Another differentiation between architectural principles is the level of coupling of the components, thus if loose coupling or tight coupling is used. Loose coupling in broader distributed system design is achieved by the use of transactions, queues provided by message-oriented middleware, and interoperability standards[1].

To achieve loose coupling in a complete way there are many characteristics that should be provided by a system and the components that this system includes. Some of them are the following, as suggested by [6]

- physical connections via mediator,
- asynchronous communication style,
- simple common types only in data model,
- weak type system,
- data-centric and self-contained messages,
- distributed control of process logic,
- dynamic binding (of service consumers and providers),
- platform independence,
- business-level compensation rather than system-level transactions,
- deployment at different times,
- implicit upgrades in versioning.

The usage of an Enterprise Service Bus (ESB) middleware became a popular way to achieve many of the desired characteristics of loose coupling. However, over engineered and mispositioned ESBs can also have the contrary effect and create undesired tight coupling and a central architectural hotspot[2].

For this reason, in ANASTACIA we try to combine the desired characteristics of the different approaches and create an integration that is based on both direct communications between components (Star architecture) and asynchronous, loosely-coupled integration using a common message broker that is scalable by design[3] (Apache Kafka). It is important to clarify that this approach helps us on achieving characteristics of event-driven architecture and enable ANASTACIA for the proper supporting of production, detection, consumption of, and reaction to events.

## 2.2 ANASTACIA INTEGRATED FRAMEWORK ARCHITECTURE

### 2.2.1 The Envisioned Platform

An important role for the decision regarding the architectural approach followed in ANASTACIA was the clarification of the platform vision regarding the way that the ANASTACIA as whole will be used. Based on the analysis of initial requirements and the use cases reported in Deliverable D1.2, in Deliverable D1.3 five main activities to be supported by the platform were identified, with each of them utilizing specific components. For the integration planning it is important to clarify how each of these components interconnects to achieve these identified activities that are shortly presented below;

- **Security policy set-up activity**. This is the initial process triggered once a security policy has been defined by the user. In this process the policy has to be configured in the platform in order to be enforced. The interpretation of the security policy claims, the configurations required to monitor the security controls associated to a policy or the definition of thresholds to identify policy violations, are some activities carried out by this process.

[1] Pautasso C., Wilde E., Why is the Web Loosely Coupled? - http://www2009.eprints.org/92/1/p911.pdf
[2] http://bulgerpartners.com/how-loosely-coupled-architectures-are-helping-the-modernization-of-legacy-software/
[3] https://www.confluent.io/blog/apache-kafka-for-service-architectures/

- **Security policy orchestration activity**. Once the policy has been defined, it is necessary to enforce the controls specified within the policy. To orchestrate the selected IoT/SDN/NFV-based security enablers, appropriate interactions with the relevant management modules are required.
- **Security monitoring activity**. In this process the monitoring information is extracted from the devices through monitoring agents and according to the security controls interpreted from the security policy. In this activity, the monitoring data is filtered and aggregated in order to carry out its analysis and the detection of anomalies.
- **Security reaction activity**. In this process the detected anomalies are evaluated to design counter measures in order to mitigate the effects of attacks and potential threats.
- **Dynamic security and privacy seal creation activity**. In this process, relevant information about detected threats, monitored information is evaluated to create a seal that determine the level of security guaranteed/offered by an IoT platform.

The aforementioned activities, along with their sub-activities and resulting architecture of the platform are described with detail to the deliverable D1.3[4] . In this deliverable the architecture will be presented in order to identify and provide technical details about the needed integration points between the components that will allow the platform integration.

### 2.2.2 Integration Points

What is extremely important for the integration activities is the identification of the integration points. Due to this importance, the needs for interfaces between the components have been identified early and there has been effort on the concrete description of the interfaces in parallel with the architecture discussions. For this reason, the main interfaces have been identified in D1.3. However, in this deliverable we provide updates (MDR interface) and description of the interfaces with technical details that would lead to the realization of the ANASTACIA architecture by a unified platform.

Figure 1 shows the ANASTACIA architecture which includes the interfaces between modules, and then more details are provided for each of these interfaces.

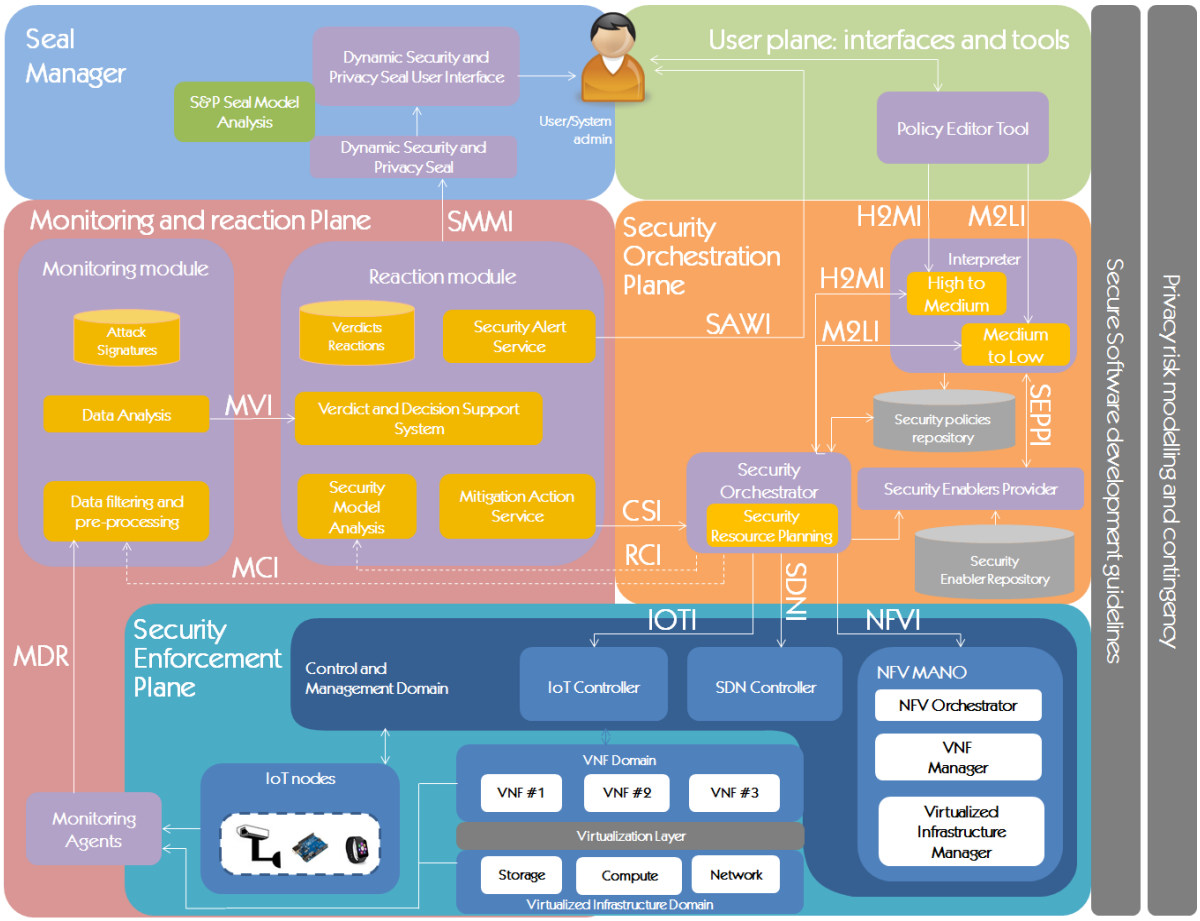



Figure 1. ANASTACIA architecture – Interface View

More specifically the interfaces introduced that were introduced in D1.3 and extended in this deliverable are the following:

- **High to Medium interface (H2MI):** Interface between the User Plane and the Orchestration Plane used for translating and refine policies. H2MI provide information at a high level of granularity. This interface is also used internally by the Security Orchestrator to get details about the capabilities that needs to be enforced within the IoT platform.
- **Medium to Lower interface (M2LI)**: Interface between the User Plane and the Orchestration Plane used for translating and refine policies. M2LI provides a lower level of granularity than the information provided by H2MI. This interface is also used internally by the Security Orchestrator to get details about the capabilities that needs to be enforced within the IoT platform.
- **MSPL Reception Interface (MRI):** This interface is used by the policy interpreter to send the MSPL to the Security Orchestrator.
- **Monitoring Configuration Interface (MCI**): This interface is used from the security orchestrator in order to configure monitoring parameters.
- **Reaction Security Configuration Interface (RCI)**: Interface between the Orchestration plane and the Monitoring and Reaction planes, used for the configuration of monitoring and reaction activities.
- **IoT-oriented Security Enforcement Plane Interface (IOTI):** This interface is used from the security orchestrator in order to configure the IoT controller.
- **SDN-oriented Security Enforcement Plane Interface (SDNI):** Interfaces between the Security Orchestrator and the SDN controllers. It provides the connectivity required among the Network Virtual Functions, and some basic security reactions.

- **NFV-oriented Security Enforcement Plane Interface (NFVI)**: This interface allows managing the security VNFs via the ETSI-oriented NFV MANO modules. The Security Orchestrator can request the enforcement of the security VNFs according to the configurations generated by the policy refinement process.
- **Security Alerts and Warnings Interface (SAWI)**: Interface between the Reaction module and the user plane which is used for the notification to the User/System admin about relevant information regarding alarms, countermeasures, etc.
- **Countermeasures Suggestions Interface (CSI)**: Interface between the Reaction module and the Orchestrator to exchange information about the countermeasures to be enforced in the IoT platform in order to react to certain incident.
- **Monitoring Verdicts Interface (MVI)**: Interface between the Monitoring module and the Reaction module used for exchanging information about detected incidents.
- **Security Enabler Provider Plugin Interface (SEPPI)**: Interface exposed by the Security Enablers Provider. It is used to get an appropriate enabler plugin during the lower policy refinement done at the Policy Interpreter, as well as providing the list of available security enablers.
- **Seal Manager Metadata Interface (SMMI)**: The interface provides the requested information to evaluate the security and the privacy in a real-time fashion. The security and privacy policies defined by the user are stored inside the policies repository and an interface is available to retrieve and set them from the seal manager.

In the following section, the detailed technical description of all the identified interfaces is presented based on bilateral and general discussions between the technical partners.

## 2.3 Detailed Description of the Interfaces

This section gathers information about the interfaces required for the implementation of the integrated solution of ANASTACIA by defining the communication between the components created in WP2-3-4-5.

The following subsections describe these interfaces (organized per activity) by detailing the following information:

- **Description**: describes the purpose of the interface
- **Component providing the interface**: describes the component that is offering the described interface.
- **Consumer components**: describes the components that are using the described interface.
- **Type of interface:** REST, XML-RPC, GUI, Java API etc.
- **Input data**: describes how data that is required by the described interface (e.g.: Methods or Endpoints, values and parameters of the interface)
- **Output data**: describes the data that is returned by the described interface (e.g.: the returned data of methods or REST call)
- **Constraints:** Any security or authentication related topics regarding this interface, specifically the need to use a secure transfer protocol. Also, any other constraints (e.g. specific prerequisites, data-types, encoding, transfer rates) which apply to the interface.
- **State:** Synchronous/Asynchronous, Stream
- **Responsibilities:** Partner that is responsible for the implementation and usage of the interface

We have to mention that the the identified interfaces, their methods and parameters are part of ongoing development and collaboration between partners, so future changes could be required. The final version of the interfaces will be documented in deliverable D6.4 - Final Technical integration and validation Report.

### 2.3.1 Interfaces for Policy Set-up Activity

The following tables describe the interfaces involved in the set-up of a new policy, comprising the interpretation of a security policy set-up at the editor, involving the interfaces H2MI (Table 1), M2LI (Table 2), SEPPI(missing reference to table), and the configuration of the monitoring and reaction modules which involve interfaces MCI (Table 9) and RCI (Table 5). These tables extend and update the information gathered in D1.3.

#### 2.3.1.1 High to Medium Interface

Table 1. Policy Editor Tool -> Interpreter H2M (H2MI)

| **High to Medium interface (H2MI)** | | | |
|---|---|---|---|
| **Description** | The interface allows requesting a policy refinement from a High level Security Policy (HSPL) to a Medium level Security Policy (MSPL), as well as to request a policy enforcement from a HSPL (avoiding to manually request M2L and MRI interfaces). | | |
| **Component providing the interface** | Policy Interpreter | | |
| **Consumer components** | Policy Editor Tool | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | h2mrefinement<br>h2menforcement | JSON data with the HSPL policy, codified in XML. A suitable list of enablers could be also provided. | JSON data with the MSPL policies and a list of candidate security enablers.<br>Policy enforcement result. |
| **Constraints** | Notice the JSON data parameters could contains more than the policy and the enablers, like matching between devices address and its human readable names. | | |
| **Responsibilities** | ○ UMU | | |

### 2.3.1.2 Medium to Lower Interface

Table 2. Security Orchestrator -> Interpreter M2L (M2LI)

| Medium to Lower interface (M2LI) | | | |
|---|---|---|---|
| **Description** | The interface allows to request a policy refinement from a Medium Level Security Policy (MSPL) to a specific enabler configuration/task | | |
| **Component providing the interface** | Policy Interpreter | | |
| **Consumer components** | Policy Editor Tool<br>Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | m2ltranslate | JSON data with MSPL policy codified in XML and the enabler name to enforce it. | Enabler´s specific Security control configuration/Task. |
| **Constraints** | M2LI uses the SEPPI interface in order to obtain the enabler plugin and performs the M2L translation. | | |
| **Responsibilities** | ○ UMU | | |

### 2.3.1.3 MPSL Reception Interface

Table 3. Policy Interpreter-> Security Orchestrator

| MSPL Reception Interface (MRI) | |
|---|---|
| **Description** | This interface can be used by the policy interpreter to send the MSPL to the Security Orchestrator. |
| **Component providing the interface** | Security Orchestrator |

| | | | |
|---|---|---|---|
| **Consumer components** | Policy Interpreter | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | Load_MSPL | JSON data with MSPL policy codified in XML and the candidate security enablers. | Acknowledgement of the reception. |
| **Constraints** | None | | |
| **Responsibilities** | ○ AALTO<br>○ UMU | | |

### 2.3.1.4 Security Enabler Provider Plugin Interface

Table 4. Interpreter -> Security Enabler Provider (SEPPI)

| **Security Enabler Provider Plugin Interface (SEPPI)** | | | |
|---|---|---|---|
| **Description** | The interface allows requesting for a plugin which implements the MSPL to Enabler translation. | | |
| **Component providing the interface** | Security Enabler Provider | | |
| **Consumer components** | Policy Interpreter | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |

| | | | |
|---|---|---|---|
| | getplugin<br><br>getenablers | Enabler´s name<br><br>Identified capabilities | Enabler´s MSPL->Security control translator software package (Software package which implements the MSPL to Lower translation)<br>A list of candidate Enablers. |
| **Constraints** | The software must implement the method getConfiguration() | | |
| **Responsibilities** | ○ UMU (getplugin)<br>○ THALES (getenablers) | | |

### 2.3.1.5 Reaction Security Configuration Interface

Table 5. Orchestrator -> Reaction definition (RCI)

| **Reaction Security Configuration Interface (RCI)** | | | |
|---|---|---|---|
| **Description** | This interface allows the Security Orchestrator to provide the Security Model-related data to the Reaction Module. In general terms, this information will be composed by the Capabilities of the Security Policy and the available countermeasures on the network to react to a detected security issue. | | |
| **Component providing the interface** | Security Model Analysis (Reaction Module) | | |
| **Consumer components** | Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | setConfiguration | The list of enforcing capabilities and list of available countermeasures. | Simple acknowledgement in the reception of the request (e.g., HTTP response status code) |
| **Constraints** | Once the Security Orchestrator has identified the capabilities expressed in the security policy, the Security Orchestrator will inform the Reaction module about these capabilities in order to correctly configure the countermeasures assessment | | |

| | |
|---|---|
| | process. The feedback received from the Security Orchestrator might also be used to provide enhanced information to the Seal Management Plane of the ANASTACIA platform. |
| **Responsibilities** | ○ THALES<br>○ AALTO<br>○ UTRC |

### 2.3.2 Interfaces for Policy Orchestration and Enforcement

The following interfaces are used for the enforcement of security policies in IoT devices. Three possible ways of orchestrating or enforcing a policy can be used depending on the interface used:

- Policy enforcement using SDN controllers through the SDNI (Table 6)
- Policy enforcement using NFV-MANO modules through the NFVI (Table 7)
- Policy enforcement using IoT controllers through the IOIT (Table 8)

#### 2.3.2.1 SDN-oriented Security Enforcement Plane Interface

Table 6. Security Orchestrator <-> SDN controllers (SDNI)

| **SDN-oriented Security Enforcement Plane Interface (SDNI)** | | | |
|---|---|---|---|
| **Description** | This interface allows managing the SDN networking configuration via the SDN controller(s). The Security Orchestrator can request the enforcement of the SDN traffic flow rules received as outcome of the policy refinement process. | | |
| **Component providing the interface** | SDN controller(s) : ONOS | | |
| **Consumer components** | Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | Flow_dropping<br>Flow_mirroring<br>Flow_forwarding | JSON with the list of parameters required to manage the flows | JSON with method execution results |

| | |
|---|---|
| **Constraints** | Regarding ONOS north-bound APIs, authentication based on user and password is required for issuing commands. Additional security features can be enabled. |
| **Responsibilities** | ○ AALTO |

### 2.3.2.2 NFV-oriented Security Enforcement Plane Interface

Table 7. Security Orchestrator <-> NFV MANO modules (NFVI)

| **NFV-oriented Security Enforcement Plane Interface (NFVI)** | | | |
|---|---|---|---|
| **Description** | This interface allows to manage the security VNFs via the ETSI-oriented NFV MANO modules. The Security Orchestrator can request the enforcement of the security VNFs according to the configurations generated by the policy refinement process. | | |
| **Component providing the interface** | NFV MANO (Management and Orchestration) modules: OSM (under evaluation) | | |
| **Consumer components** | Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | Onboard/export Virtual Network Function Descriptor (VNFD)/ Network Service Descriptor (NSD)<br>Create/Delete Network Service (NS)<br>Execute configuration primitives on Network Services. | Data packages defining NSD/VNFD.<br>Information about the NS to manage | Method execution results |
| **Constraints** | OSM authentication is based on user and password is required for issuing commands. Also, HTTPs is enabled. Additional security features can be considered. | | |

| | |
|---|---|
| **Responsibilities** | ○ AALTO<br>○ THALES |

### 2.3.2.3 IoT-oriented Security Enforcement Plane Interface

Table 8. Security Orchestrator <-> IoT controllers (IOTI)

| **IoT-oriented Security Enforcement Plane Interface (IOTI)** | | | |
|---|---|---|---|
| **Description** | This interface allows managing the configuration of IoT nodes via specific IoT controllers. The Security Orchestrator can request the enforcement of the security controls within the IoT nodes according to the configurations generated by the policy refinement process. | | |
| **Component providing the interface** | IoT controllers | | |
| **Consumer components** | Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | device/device_id<br>bootstrapping | iot_resource<br>PEMK key | IoT resource values/result of the operation (turn off, disable radio, bootstrapping), in plain text |
| **Constraints** | The PEMK (PaC-EP Master Key) key could be acquired previously through an AAA architecture. | | |
| **Responsibilities** | ○ UMU/OdinS | | |

## 2.3.3 Interfaces for Monitoring

The following tables describe the interfaces involved in the Monitoring processes. First, the configuration of the configuration of the monitoring is provided by MCI interfaces (Table 9) and the collection of monitoring data is described in the Monitoring Data Receiver (MDR) set of interfaces (Table 10) .

### 2.3.3.1 Monitoring Configuration Interface

Table 9. Orchestrator -> Monitoring definition (MCI)

| Monitoring Configuration Interface (MCI) | | | |
|---|---|---|---|
| **Description** | This interface allows configuring the Monitoring Module from the Security Orchestrator. It is intended to provide the required parameters to refine the detection of potential threats on the network. | | |
| **Component providing the interface** | Data Filtering Component (Monitoring Service) | | |
| **Consumer components** | Security Orchestrator | | |
| **Type of Interface** | REST | | |
| **State** | Synchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | setAnalysisParams | The list of enforcing capabilities and enforced security policies. | Simple acknowledgement in the reception of the request (e.g., HTTP response status code) |
| **Constraints** | Once the Security Orchestrator has identified the capabilities expressed in the security policy, the SO will inform the Monitoring module about these capabilities in order to correctly configure the monitoring agents | | |
| **Responsibilities** | ○ UBITECH<br>○ AALTO | | |

### 2.3.3.2 Monitoring Data Receiver

Table 10. Monitoring Agents-> Monitoring Module (MDR)

| Monitoring Data Receiver (MDR) | |
|---|---|
| **Description** | This integration point is needed in order to allow the Monitoring Agents to provide their output to the Monitoring Module through the Data Filtering Component. It is not a single interface but a collection of Kafka topic[4] that will be used for the collection of the data from a diverse set of monitoring agents that have or will be |

[4] https://kafka.apache.org/documentation/

| | | | |
|---|---|---|---|
| | developed in the project duration. A Kafka topic is where data are published to by a producer (entity creating data) and pulled from a consumer (entity consuming data) | | |
| **Component providing the interface** | Kafka Message Broker | | |
| **Consumer components** | Data Filtering Component (Monitoring Service) | | |
| **Type of Interface** | Type of Kafka Topic depends of the Monitoring Agent | | |
| **State** | Asynchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | One topic per each monitoring agent will be used | N/A (topic values depended on the monitoring agents) | N/A (topic values depended on the monitoring agents) |
| **Constraints** | Each monitoring agent should be able to connect to the Kafka Broker | | |
| **Responsibilities** | ○ UBITECH<br>○ Partners providing monitoring agents (for creating the needed producer) | | |

## 2.3.4 Interfaces for Reaction Activity

The following interfaces are used for exchanging relevant data required for the fulfilment of a security policy within an IoT platform. This includes:

- The notification of detected incidents between the Monitoring and the Reaction modules through the MVI (Table 11)
- The notification of alerts and countermeasures from the Reaction module to the User/System admin through the SAWI (Table 12)

### 2.3.4.1 Monitoring Verdicts Interface

Table 11. Monitoring -> Reaction definition (MVI)

| **Monitoring Verdicts Interface (MVI)** | |
|---|---|
| **Description** | This interface is intended to provide the required monitoring information from the Monitoring to the Reaction Module. The transferred data is mainly composed of the verdicts of the security properties tested on the network. |
| **Component providing the interface** | Verdict and Decision Support System (Reaction Module) |

| | | | |
|---|---|---|---|
| **Consumer components** | Data Analysis (Monitoring Module) | | |
| **Type of Interface** | RPC or REST | | |
| **State** | Asynchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | backlogBolt | JSON including: alarmID, BacklogID, AlarmEvent. Alarm event containing related to the incident detected. | None |
| **Constraints** | None | | |
| **Responsibilities** | ○ MONT<br>○ ATOS | | |

### 2.3.4.2 Security Alerts and Warnings Interface

Table 12. Reaction -> User/System Administrator definition (SAWI)

| **Security Alerts and Warnings Interface (SAWI)** | |
|---|---|
| **Description** | This interface will transfer the alerts and warnings from the Reaction Module to the end-user interfaces. It is designed as a communication channel between the Reaction Module and the ANASTACIA User Plane. |
| **Component providing the interface** | Security Alert Service (Reaction Module) |
| **Consumer components** | End-user interface |
| **Input data / Output Data** | Several options were considered;<br>1) Database (relational, or noSQL, to be evaluated in function of the data flow), directly accessed from the user plane interfaces<br>2) JSON/XML format passed to the user plane<br>3) Syslog format passed to the user plane |

| | | | |
|---|---|---|---|
| | The first option was chosen in order to avoid complexity issues, but the need for the other options will considered during the development of both reaction model and the user plane components. | | |
| **State** | Asynchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | Specific tables on the database containing the information | The set of detected security issues (for raiseAlert), and the applied countermeasures (for informReaction) | None |
| **Constraints** | None | | |
| **Responsibilities** | ○ ATOS<br>○ MONT<br>○ CNR<br>○ UTRC | | |

### 2.3.4.3 Countermeasures Suggestions Interface

Table 13. Reaction -> Orchestrator definition (CSI)

| **Countermeasures Suggestions Interface (CSI)** | | | |
|---|---|---|---|
| **Description** | This interface was conceived to transmit the set of suggested countermeasures from the Reaction module to the Security Orchestrator | | |
| **Component providing the interface** | Security Orchestrator | | |
| **Consumer components** | Mitigation Action Service (Reaction Module) | | |
| **Type of Interface** | REST (MSPL) | | |
| **State** | Asynchronous | | |
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |

| | | | |
|---|---|---|---|
| | informCountermeasures | A list of the possible countermeasures to implement | A MSPL based file containing details on mitigation actions to be deployed with all required information. Some information can be the devices affected (IPs, subnet) the attack to mitigate, and other information that is currently being defined. |
| **Constraints** | Following the usage of open standards, this interface is intended to use the OpenC2 format to inform the set of countermeasures that will be applied on the network. High-priority, secure and reliable communication is considered important for this interface. | | |
| **Responsibilities** | ○ AALTO<br>○ ATOS | | |

## 2.3.5 Interfaces for Seal Creation

### 2.3.5.1 Seal Manager Metadata Interface

The following interface SMMI (Table 14) is used for the exchange of the relevant data that the seal manager needs in order to create the Dynamic Security and Privacy Seal.

Table 14. Reaction -> Seal Manager definition (SMMI)

| **Seal Manager Metadata Interface (SMMI)** | |
|---|---|
| **Description** | The interface provides the requested information to evaluate the security and the privacy in a real-time fashion. The security and privacy policies defined by the user are stored inside the policies repository and an interface is available to retrieve and set them from the seal manager. |
| **Component providing the interface** | Dynamic Security and Privacy Seal |
| **Consumer components** | Security Alert Service, Security Model Analysis |
| **Type of Interface** | Usage of STIX/TAXII (Structured Threat Information Expression and Trusted Automated eXchange of Indicator Information). Alternative options considered are he direct access to mysql database or is using Kafka broker and JSON format |
| **State** | Asynchronous |

| | | | |
|---|---|---|---|
| **Input data / Output Data** | **Methods or endpoints** of the interface | **Parameters** of the method | **Return Values** of the method |
| | computeSecuritySeal | TO BE DEFINED | none |
| **Constraints** | TO BE SPECIFIED | | |
| **Responsibilities** | ○ ATOS<br>○ MAND<br>○ MONT | | |

## 2.4 TECHNICAL INTEGRATION MECHANISMS AND PROCESS

As it was shown in the architecture diagram of Figure 1 and in the explained integration approach, ANASTACIA Platform consists of (1) components responsible for processing input data provided by the monitored streams, (2) components that are responsible for making decision and reaction and (3) components that provide useful information to the end user. All these components are integrated both using direct communications between components (Star architecture) and asynchronous, loosely-coupled integration using a common message broker when it is needed.

Table 15. ANASTACIA integration mechanisms

| Multiple Facets Integration | |
|---|---|
| **At Deployment Level** | **Configuration of components' deployment using Docker compose**<br>**Dedicated container registries using Gitlab** |
| **At Interfaces Level** | **Documentation of Interfaces using Swagger** |
| **At Code Level** | **Dedicated code repositories using Gitlab** |
| **At Knowledge Level** | **Dedicated folder for collaboration on the shared repository of consortium**<br>**Usage of Slack** |

For the technical integration in ANASTACIA we need many different components to be deployed and communicate using dedicated interfaces or the common message broker. For achieving this we are using Docker Compose[5] files. Docker Compose is a tool for defining and running multi-container Docker applications, based on a YAML file that is used to configure application's services. Then, with a single command, all services can be created and started using the configuration file. Following this approach doesn't mean that all components of ANASTACIA should be containerized and deployed using Docker. Some services may still be provided from a centralized point or through dedicated physical or virtual machines. What is managed through Docker compose configuration file (and by respecting some development guidelines) is the way that all components are possible to be deployed and communicate.

In order to make the whole integration flow to work based on Docker Compose in an autonomous and continuous way for ANASTACIA, we will try to create Docker based container images for the components developed, whenever this possible. In comparison to virtual machine that needs to include infrastructure

[5] https://docs.docker.com/compose

configuration and the whole OS, the containers image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files. A container is a runtime instance of an image—what the image becomes in memory when actually executed. In comparison to a Virtual Machine (VM) that is completely isolated, a container is partially isolated from the host environment, as it uses the kernel calls and commands of the host OS, but accessing host files and ports is possible only if configured to do so.

Images are created using Docker and are possible to be configured using Dockerfile. A Dockerfile contains instructions on how to create the desired image based on pre-existing images. More information regarding the creation process is provided in section 2 that follows.

The pre-existing images can be stored and retrieved from image repositories called Docker registries. Such a Docker Registry is used for ANASTACIA and is a stateless, highly scalable server-side application that allows storing and distributing Docker images. It works similar to Git, as collaborators can login and then push or pull the images that they or other partners are creating.

The configuration of the multiple available components of the platform that are communicating through the interfaces (mostly REST) can be helped with Docker compose files, through the usage of Environmental Variables for configuration. Most important parameters that are usually needed for this action are service urls, ports or any other information needed for the acknowledgment and configuration between services. With environmental variables existing in each Docker compose based deployment, the service developer shall utilize the available variables at code level in order to avoid the hardcoding of parameters that make applications difficult to deploy and, in the end, develop and test a project with many different partners in remote locations, like ANASTACIA.

### 2.4.1 Using Docker for Integration

Dockerizing an application is a very diverse procedure and multiple approaches can be followed, for this reason dedicated guidelines and examples have been shared among the technical partners.

First step for collaborative working was that each partner needs to join the ANASTACIA project page in GitLab[6], that will be used for the hosting of private code and container repositories, as well tools for the support of CI activities. For hosting integration related material of the whole framework, as docker-compose.yml files or the needed docker images[7], a project called framework has been created.

#### 2.4.1.1 Creating Docker Images for components

An application may be “dockerized” by using a variety of approaches. For the full scope of possibilities refer to the official documentation also given in the appendix. Here we will cover the most common guidelines:

An application may depend upon multiple components. A LAMP stack for example needs at the minimum a MySQL and Apache Web Server. A single container can contain the whole stack, or a separate container for each component may be used. Using a single container is generally easier, while using multiple containers is cleaner. Each partner may dockerize components as needed, with some basic suggestions like using separate containers for databases, web applications and web servers. If a lightweight server is used a single container can be used, like the case of Spring boot applications.

For dockerizing an application, a Dockerfile is needed. The Dockerfile contains instruction on how to construct the instance of an image (called container). The Dockerfile contains directives like:

[6] https://gitlab.com/anastacia-project

[7] registry.gitlab.com/anastacia-project/framework

- defining a base image to be used (e.g. Ubuntu 16). ANASTACIA developer can login and use ANASTACIA registry order to re-use the images already created as base images, by using the command (FROM *registry.gitlab.com/ANASTACIA-project/framework/<image_name>:tag*)
- adding local files to the file system of the container
- commands to be executed upon initialization of the container (e.g. packages to be installed)
- Ports to be exposed
- Commands that launch the applications

#### 2.4.1.2 ANASTACIA Docker Registry usage

A Docker registry has been setup for the purposes of ANASTACIA development using Gitlab. Developers that want to push an image to the repository should first tag it with the repository and then push it using the following commands.

```
$ docker login registry.gitlab.com
$ docker build -t registry.gitlab.com/ANASTACIA-project/framework/<image_name>:<tag> .
$ docker push registry.gitlab.com/ANASTACIA-project/framework/<image_name>:<tag>
```

where:

- *image_name*: the name of the image. Typically, this is the same as the local image name.
- *tag:* Optional as parameter but need to properly support the continuous integration workflow. It is used for creating versions of the same image. For the first iteration version 0.1 will be used as tag for all images. Also, if you not specify the tag, the docker will automatically set the tag latest.

Similarly, an image can be pulled by executing:

```
$ docker pull registry.gitlab.com/ANASTACIA-project/framework/<image_name>:<tag>
```

### 2.4.2 Integration at Interface Level

For the better coordination of the development of the interfaces that are used by the Interfaces we can use Swagger. This way each partner will be responsible to provide the appropriate documentation for the interface usage and this documentation will be generated automatically in order to allow the consumer of the interface to edit/adapt to the REST body changes.

As seen in the following Figure 2, Swagger is easily added at the code of the developed service. It supports all major languages, and in the example below of a Spring based Java application it identifies REST components automatically.

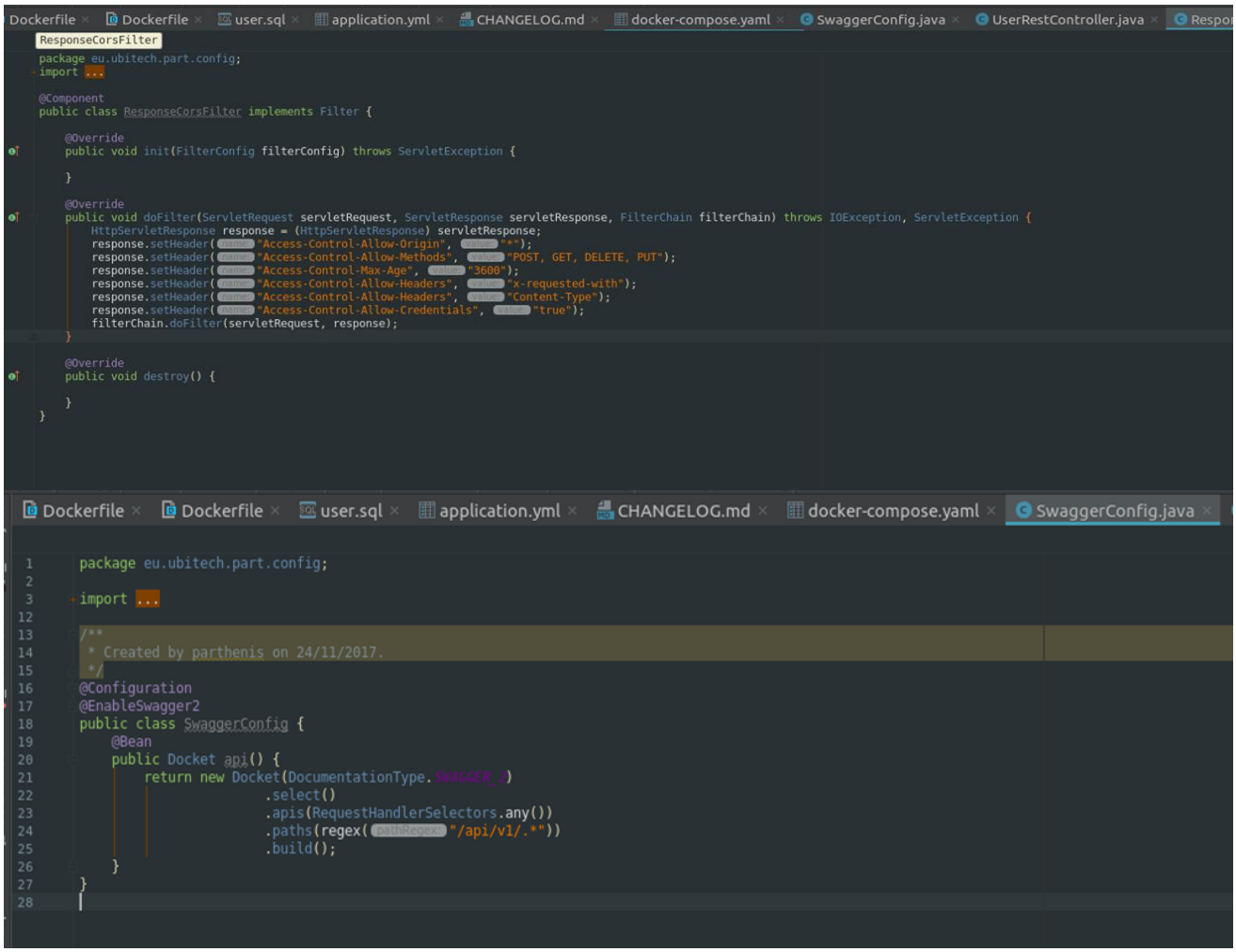



Figure 2. Sample Configuration of Swagger on a Spring boot application

Swagger then produces the appropriate JSON for the documentation. Based on this JSON, a dedicated UI can render nicely the whole documentation, while it is also possible to create client libraries for most major languages.

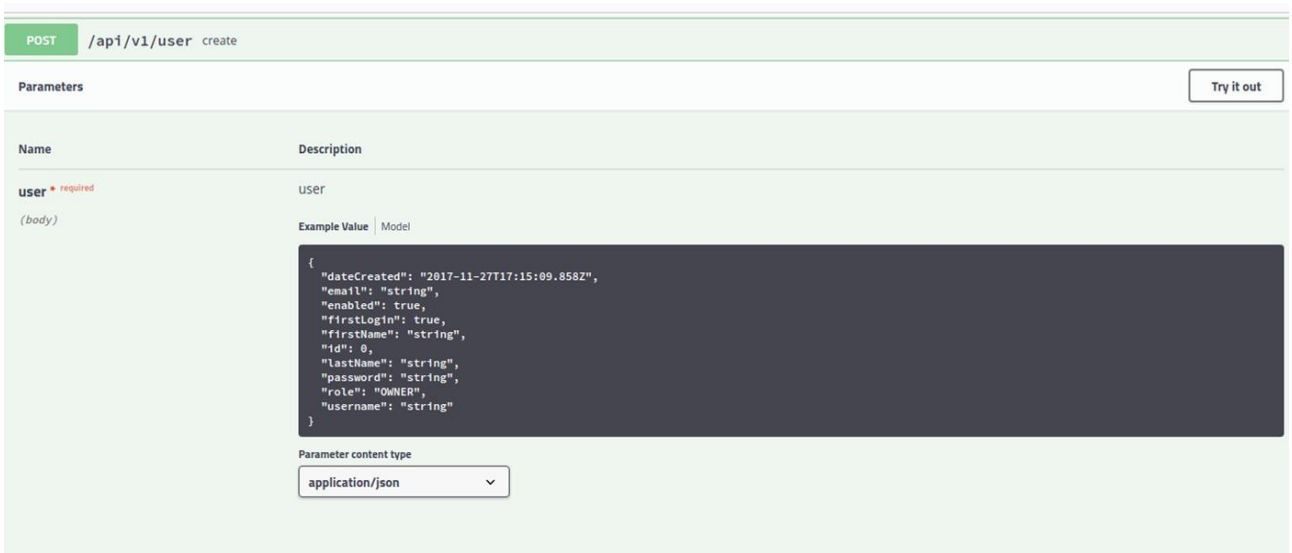



Figure 3. Sample of Swagger UI

Swagger UI is deployed as part of the development, production or use case deployments using the following code in the Docker-compose file.

```
swagger:
  container_name: "ANASTACIA_swagger"
  image: swaggerapi/swagger-ui
  environment:
    API_URL: "http://localhost:8080/v2/api-docs"
  ports:
   - 8070:8080
```

#### 2.4.2.1 Asynchronous Operations

For interfaces that need asynchronous operation mode for their communication, a message broker can be used. The publish/subscribe (pub/sub) messaging pattern is realized using destinations known as topics. Publishers send messages to the topic and subscribers register to receive messages from the topic. Any messages sent to the topic are automatically delivered to all subscribers. In ANASTACIA we are using Apache Kafka[8] as message broker for the integration of monitoring agents feedback to the Data Filtering and Preprocessing component, while it might also be considered its use between components needing asynchronous communication.

### 2.4.3 Code Level Integration - Working on the same components

In the cases that multiple partners need to work on the same components, code level integration is supported with a code repository that is available for all partners that need to work together or to store their component's code safety. The source code repositories are available at: https://gitlab.com/ANASTACIA-project

[8] https://kafka.apache.org/

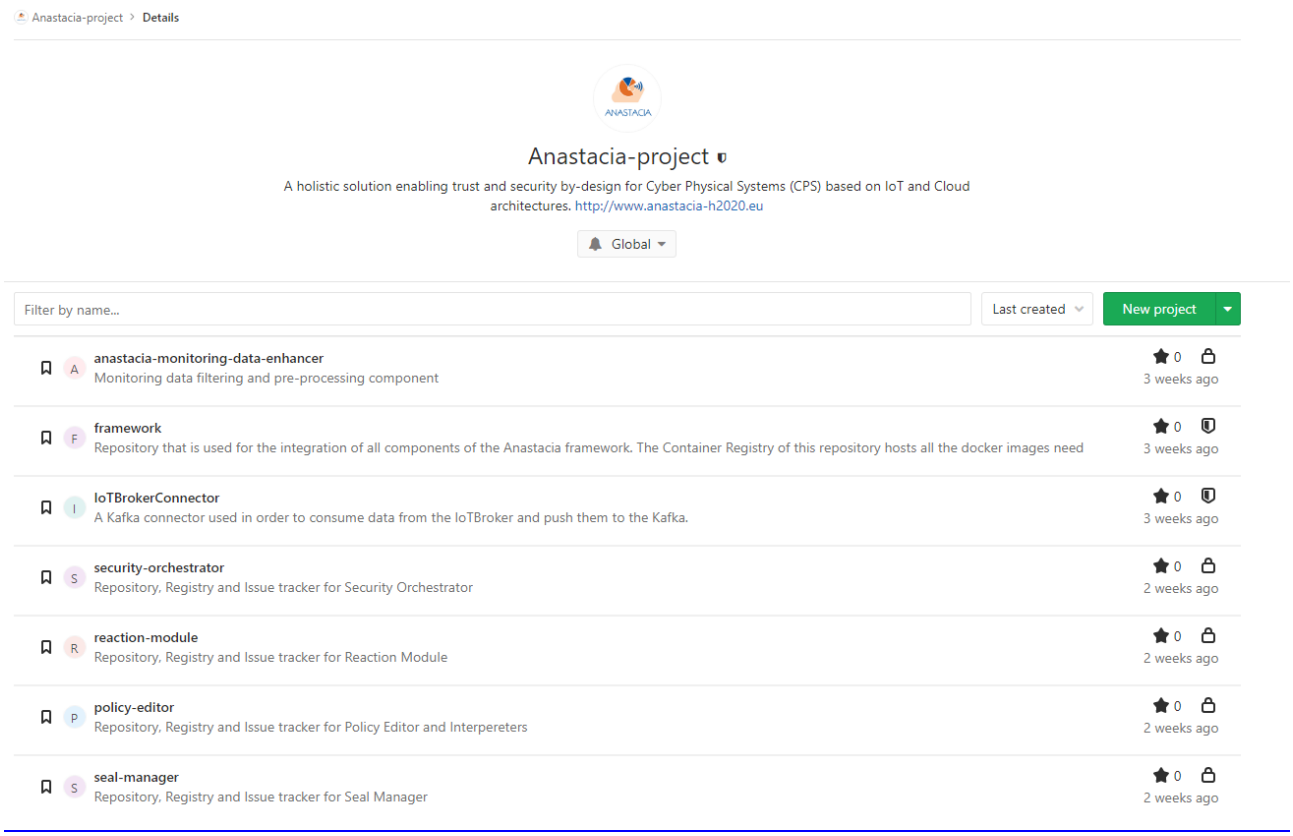



**Figure 4. ANASTACIA project group in GitLab**

### 2.4.4 Shared Knowledge

The last part to cover for the technical integration and collaboration mechanism is how to allow partners that are working in distributed manner to collaborate. This important parameter is often performed without setting strict rules, but in the case of ANASTACIA we will try to collect the shared knowledge in order to ease the development and integration. Towards these directions the following steps have been performed. Initially a slack [9] team for ANASTACIA has been created, and dedicated channels for each work package have been created. Then, in order to achieve integration planning goals, a document collecting all the details of the technical integration, together with instructions and examples had been created and uploaded to the common repository of the project. This document will be constantly updated when needed. Finally, with GitLab it is also possible to create wiki pages for each of the component in order to allow partners developing a component to provide the needed information and instructions in a structured format.

[9] https://slack.com/

# 3 PLATFORM IMPLEMENTATION AND INTEGRATION PLANNING

In ANASTACIA we perform two cycles of technical and user evaluations during the project period, while the development of the platform and its evaluation will be performed in parallel. The results of the validation and the evaluation need to be fed back into the development cycle, improving the user experience and detecting open issues. Therefore, after the evaluation will be performed, the resulting issues need to be discussed by the consortium and the list of functional requirements for the next software development iteration will be updated accordingly.

As depicted in Figure 5, the development and testing lifecycle combines the V-model[10] process with short, concurrent development cycles. Component testing is performed frequently, and integration testing that depends on the interfaces of two or more partners is performed occasionally, and specifically when changes on interfaces are done. Due to the effort and cost of full system tests and user testing, these evaluations must be scheduled with a lower frequency than component and integration tests. Also, they are performed independently of the shorter component development cycles, based on a predefined release plan.

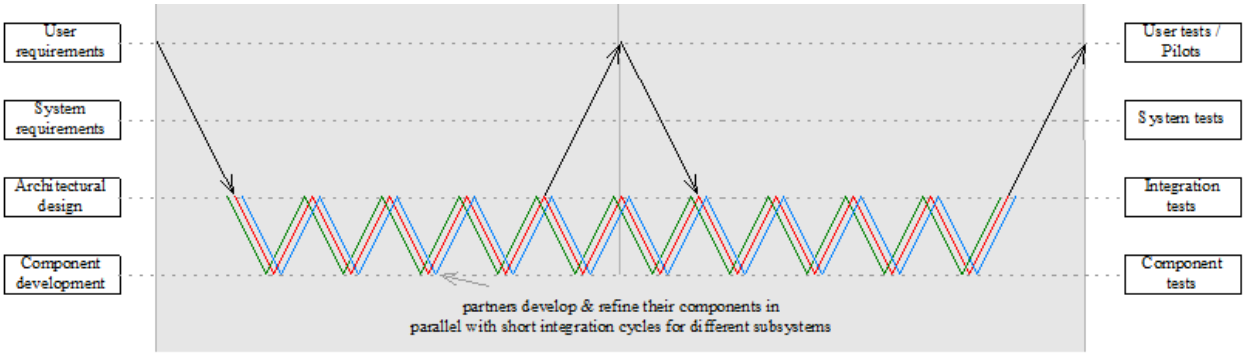



Figure 5. The ANASTACIA development lifecycle combines the V-model with short, concurrent development cycles

For the whole platform development, a two-cycle process will be followed: a first cycle will be performed with the initial requirements definition, the design and development of first demonstrators, and the first tests performed by the end users. A second cycle will follow with the revisions based on the users' feedback, the design and development of the final demonstrators and the final tests and evaluations of end users and final users[3] .

However, smaller iterations with changes on the architectural design and the development will be used through the project duration. This continuous process will also be documented with revised versions of the official reports for requirements analysis (D1.4 due on M23) and architecture (D1.5 due on M36).

## 3.1 MULTI-ITERATION/RELEASE PLAN

Based on the plan of having to two releases of ANASTACIA framework, the consortium had to make decisions regarding the specific functionalities that will be supported on each release, in order to coordinate the development, iteration and testing process. Meanwhile the integrations between components has starter in order to allow the iterative implementation of the platform and assure the delivery of the first version on M19.

The final version of the integrated platform will be delivered on M36 and will include all the individual functionalities per components and extend the integration of the first version to support the functionalities provided in the final version of the components.

[10] https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html

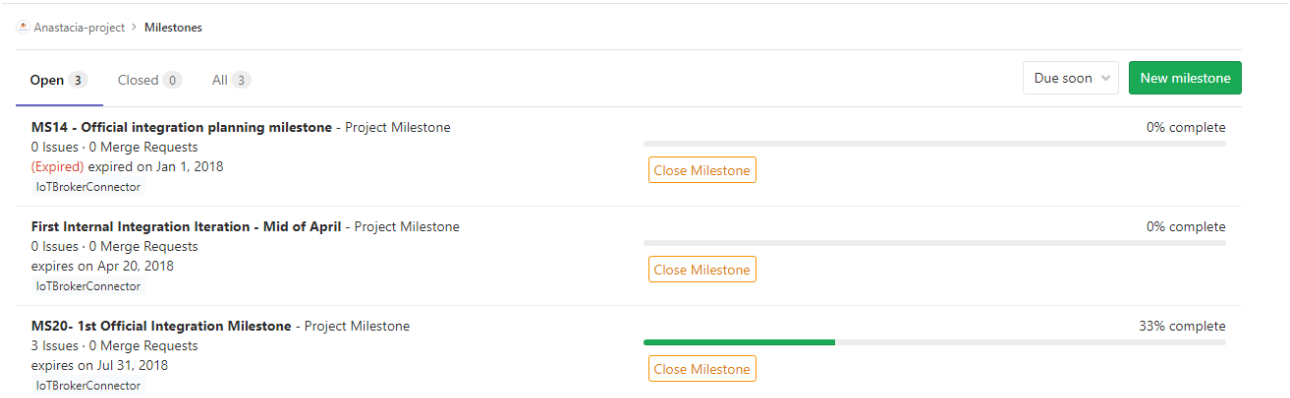



Figure 6. The ANASTACIA Milestones as part of the development plan

### 3.1.1 1st Platform Release and Validation Iteration

The first release of ANASTACIA is dictated by the Milestone MS20 that is due on M19 and the goal is to complete the "first cycle of integration and technical validation". After that, by the end of M20 and according the milestone "MS23- First iterative cycle of development completed", all the validation results will be taken under account for the closure of the first cycle of development of all components, in order to start working into the second-and final- release of the platform.

Table 16. ANASTACIA components' status for first release (M19)

| Component | Responsible Partner (Subcomponent) | Status for First Release |
|---|---|---|
| **Policy Editor Tool** | UMU | First implementation covering the policy model for the first main identified use cases. |
| **Interpreter** | UMU | A first implementation comprising the H2M refinement and M2L translation processes for the first main identified use cases. |
| **Security Enablers Provider** | OdinS (DTLS proxy) | First development of DTLS proxy for enabling secure channels between IoT devices. Creation of the policy plugin for the enforcement by Orchestrator. |
| | OdinS (AAA Architecture) | First development of security mechanisms for providing authentication and authorization in the IoT networks. Creation of the policy plugins for the enforcements by Orchestrator. |
| | OdinS (Network Authenticator) | First development of Network Authenticator for enabling secure bootstrapping of IoT devices. Creation of the policy plugin for the enforcement by Orchestrator. |

| | | |
|---|---|---|
| | UMU (IPTABLES) | First development of filtering capabilities using the IPTABLES enabler. |
| | UMU(IoT Controller) | First implementation of power management, bootstrapping and IoT honeynet capabilities. |
| | UMU (SDN ONOS NB & SDN ODL NB) | First implementation of filtering and forwarding capabilities. |
| | UMU (Cooja) | First implementation of IoT honeynet translation and deployment. |
| | UBITECH (Kippo) | First implementation of the ssh honeynet enabler. |
| **Security orchestrator** | AALTO | First implementation of the Security Orchestrator incorporating the different SDN and NFV features. |
| **IoT controllers** | OdinS / UMU | First development and integration with the Orchestrator component for enabling a subset of security actuations over IoT devices .<br><br>First implementation of the IoT controller northbound/southbound endpoints in order to cope with the first main identified use cases. |
| **NVF orchestrators** | AALTO | Upgrade to Open Source Mano Release 3. (TESTING)<br><br>Definition of the relevant VNFD (Virtual Network Function Descriptor) and NSD (Network Service Descriptor) for each security appliance. |
| **SDN controllers** | AALTO | SDN driver has been updated, in order to better support the new functionalities according to the use cases. |
| **Monitoring Agents** | OdinS | Integration of IoT-Broker with Data Filtering for providing 2 different types of monitoring data such as sensor data and attacks notifications. |
| | MONT | Integration of MMT-Probe into the Kafka Broker, providing the extracted information to all Kafka subscribers |
| | ATOS | Integration with three main sources of sensors: (1) security sensors deployed in the network,(2) UTRC Data analysis tool and (3) the Montimage Monitoring Tool. Implementation of interface for the reception of logs from these three sources. Creation of plugins |

| | | |
|---|---|---|
| | | for processing the events received from these sources. Normalization of events to be correlated. |
| **Data Filtering and pre-processing Component** | UBITECH | Integration with the Data Analysis components for providing filtered output and with 3 different types of monitoring data as input; IoT devices data, Network events, security events. Aggregation of data and filtering of |
| **Data Analysis Component** | MONT | Integration of the MMT-Security Module with the Atos' XL SIEM tool, using the syslog format required by Atos' tool. |
| | ATOS | Interpretation of normalized events. Storage of the normalized events in a database, which will be available for the Security and Privacy Seal module. Correlation of events looking for security incidents. Generation of security alerts, along with the criticality of the alert. Notification of the alert to the Verdict and Decision Support System. |
| **Security Model Analysis Component** | THALES | Interpretation of reaction capabilities received from the orchestration at Reaction set-up time. |
| **Verdict and Decision Support System** | ATOS | Initial definition of the inputs for the risk assessment evaluation carried out as part of the decision support system. Definition of the criteria for the decision support. Integration of the format used for the exchange description of the reactions. |
| **Mitigation Action Service** | ATOS, MONT | Definition of a data format for the exchange of reactions information between the Reaction module and the Orchestration module. |
| **Security Alert Service** | ATOS, CNR | Initial deployment of a graphical dashboard that integrates information about events received |
| **Dynamic Security and Privacy Seal component** | MAND, DG, AS | Dynamic Security and Privacy Seal implementation partially completed |
| **Dynamic Security and Privacy Seal User Interface** | MAND, DG, AS | GUI of Dynamic Security and Privacy Seal implementation partially completed |

### 3.1.2 Final Version of the Platform and Validation Iteration

The final version of ANASTACIA Framework will be delivered at the end of the project, by M36. This version will be fully integrated and the technical validation of this final version of ANASTACIA framework will be delivered, as part of milestone "MS32 - Second iterative cycle of development completed and validated".

Table 17. ANASTACIA components' status for final release (M36)

| Component | Responsible Partner (subcomponent) | Status for Final Release |
|---|---|---|
| **Policy Editor Tool** | UMU | Final implementation covering the main identified use cases, allowing to model policies through the GUI and request their enforcement. |
| **Interpreter** | UMU | Final implementation comprising the H2M refinement and M2L translation processes for the main identified use cases. |
| **Security Enablers Provider** | OdinS (DTLS proxy) | Final integration of DTLS proxy to enable the dynamic deployment by NVF orchestrator. |
| | OdinS (AAA Architecture) | Final integration of all AAA security mechanisms to enable the dynamic deployment by NVF orchestrator. |
| | OdinS (Network Authenticator) | Final integration of Network Authenticator to enable the dynamic deployment by NVF orchestrator. |
| | UMU (IPTABLES) | Final development of filtering capabilities using the IPTABLES enabler in order to cope with the main identified use cases. |
| | UMU (IoT Controller) | Final implementation of power management, bootstrapping and IoT honeynet capabilities. |
| | UMU (SDN ONOS NB & SDN ODL NB) | Final implementation of required networking capabilities by the use cases. |
| | UMU (Cooja) | Final implementation of IoT honeynet translation and deployment as VNF, from different IoT architecture topologies. |
| | UBITECH (Kippo) | First implementation of the ssh honeynet enabler. |
| **Security orchestrator** | AALTO | Final implementation of the Security Orchestrator as a virtual instance covering the defined use cases. |

| | | |
|---|---|---|
| **IoT controllers** | OdinS / UMU | Final development and integration with the Orchestrator component for enabling all security actuations over IoT network.<br><br>Final implementation and integration of the IoT controller northbound/southbound endpoints in order to cope with the main identified use cases. |
| **NVF orchestrators** | AALTO | Final version of Open Source Mano with the relevant VNF and NS descriptors and the final version of the OSM driver. |
| **SDN controllers** | AALTO | Final version of the ONOS controller with the relevant underlying architecture and the final version of the ONOS driver. |
| **Monitoring Agents** | OdinS | Final development and integration of all attacks notifications provided by IoT broker and AAA architecture when detect malicious behaviours according to use cases proposed in ANASTACIA project. |
| | MONT | Integration of the Adapted version of the MMT-Probe to support all the IoT protocols involved in the ANASTACIA use cases. |
| | ATOS | Implementation of new plugins for the incorporation of new sources of events, including new virtual security services. |
| **Data Filtering and pre-processing Component** | UBITECH | Creation and Integration of the configuration interface (MCI) with appropriate platform mechanism that need to refine the parameters of filtering. Finalization of appropriate models used for filtering and pre-processing of data. |
| **Data Analysis Component** | ATOS | Incorporation of cross correlated alarms that integrates different events from different sources for the generation of more accurate alarms. |
| **Security Model Analysis Component** | THALES | Incorporation of new reaction capabilities included in the second stage of the project |
| **Verdict and Decision Support System** | ATOS | Complete creation of a Risk Assessment engine within the decision support system for the selection of mitigation actions. |

| | | |
|---|---|---|
| **Mitigation Action Service** | MONT | Implementation of the standard language to communicate the set of computed countermeasures to the Security Orchestrator. |
| | ATOS | Complete integration between the Orchestrator and the Reaction Module using the data structure used for the exchange of reaction information. |
| **Security Alert Service** | ATOS, CNR | Incorporation of alerts, configuration of the decision support service, visualization of reports, interfaces for the configuration of monitoring, alerting and reaction components. |
| **Dynamic Security and Privacy Seal component** | MAND, DG, AS | Dynamic Security and Privacy Seal implementation completed |
| **Dynamic Security and Privacy Seal User Interface** | MAND, DG, AS | GUI of Dynamic Security and Privacy Seal implementation completed |

# 4 PLATFORM DEPLOYMENT OVERVIEW

The deployment of ANASTACIA as platform will be executed in close collaboration with the use case partners of the project, as the platform will be tailored to the needs of each use case. However, for development and demonstration purposes a test installation of the core platform components has been provided in a cloud infrastructure using OpenStack based virtual machine software.

| Technical specifications or requirements of existing components | | | | | |
|---|---|---|---|---|---|
| **Component(s)** | Memory | Storage | Processor | Type | OS |
| **Platform Message Brokerage** | 8GB | 20GB | 2VCPU | VM hosted in OpenStack | Ubuntu 16.04 |
| **Data Filtering and pre-processing Component** | Co-hosted in the same VM with Message Broker | | | | |
| **Monitoring Atos Agents** | 8Gb max | 50Gb | A normal Intel i5 (minimum) | On premises VM hosted in VMWare or Virtualbox or hosted in a Cloud | Ubuntu based |
| **Data Analysis Component** | 16Gb min | 100Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting | Ubuntu based |
| **Verdict and Decision Support System** | 16Gb min | 100Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting (can be hosted the same VM as above) | Ubuntu based |
| **Mitigation Action Service** | 16Gb min | 100Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting (can be hosted the same VM as above) | Ubuntu based |
| **Security Alert Service** | 16Gb min | 100Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting (can be hosted the same VM as above) | Ubuntu based |
| **IoT Broker** | 16Gb min | 100Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting | Ubuntu 16.04 |
| **AAA Architecture** | 16Gb | 40Gb | Intel Xenon 2CPU | VM hosted in a Cloud hosting | Ubuntu 16.04 |

| | | | | | |
|---|---|---|---|---|---|
| **Network Authenticator** | 8Gb | 20Gb | Intel Xenon 2CPU | VM hosted in IoT network | Ubuntu 16.04 |
| **Montimage Monitoring Tool** | 4Gb | 20Gb | 2VCPUs | VM ready to be deployed in the monitored network. | Ubuntu 16.04 |
| **Security Orchestrator** | 2Gb | 20Gb | 1vCPU | VM hosted in Aalto's Openstack Cluster | Ubuntu 16.04 |
| **SDN controller (ONOS)** | 8Gb | 40Gb | 4vCPU | VM hosted in Aalto's Openstack Cluster | Ubuntu 16.04 |
| **Open Virtual Switch** | 1Gb | 10Gb | 1vCPU | VM hosted in Aalto's Openstack Cluster | Ubuntu 16.04 |
| **NFV Orchestrator (Open Source Mano -OSM)** | 8Gb | 120Gb | 8vCPU | VM hosted in Aalto's Openstack Cluster | Ubuntu 16.04 |
| **DSPS Web server** | 4-8Gb | 20Gb | 4-8VCPU | VM hosted in MI server | Ubuntu 16.04 |
| **DSPS servers** | 4Gb | 25Gb | 4VPCU | VM hosted in MI server | Ubuntu 16.04 |
| **Policy Interpreter** | 2GB | 20GB | 1vCPU | VM hosted in UMU server | Ubuntu 16.04 |

# 5 PLATFORM TESTING AND VALIDATION PLAN

This section presents the platform testing and validation as part of the overall evaluation strategy in the context of ANASTACIA. Here, the integration and testing plan will be analysed, and the actual testing and evaluation results will be provided in deliverables D6.2, D6.3. For the creation of the overall testing and validation plan analysis of standards and common methodologies has been made, like the IEEE 829[11] for the creation of test plan and the ISO 250010[12] for the identification of important attributes that we should measure for the evaluation of the overall framework and the individual components.

Therefor in ANASTACIA we define the following facets of testing:

- Unit testing that can be performed by the separate development teams when new functionalities are developed.
- Integration testing performed by the development teams in order to test the smooth co-operation between the various layers and components. The integration tests and also any unit tests that will be created for the project validation will be continuously executed based on continuous integration (CI) scheme that is documented in section 5.4.
- Stress testing can be performed for the benchmarking of the system or specific components. Due to the fact that ANASTACIA framework targets to a TRL of 5 according to project DoA [3] , stress tests will be a supplementary action that can be performed for the components that partners believe would benefit from this action.
- Validation based on the requirements' coverage (functional completeness and correctness).
- User acceptance testing will ensure the usability of the system will be performed during the use case evaluation and documented in deliverables D6.3 and D6.6.

## 5.1 UNIT TESTING

An important part of both the integration and the validation process is the execution of unit tests. Unit tests are the tool to test the functional modules of software. A suitable unit test is applied to the piece of code without any dependencies on other code parts. Therefore, the developer of the particular layers will test their components by means of unit tests before integrating them into the full application. In the case of ANASTACIA framework, development is based on the development of standalone components but also on the adaptation and integration of existing components. Therefore, unit testing at the lowest level not a primordial part of the general testing methodology of ANASTACIA, as the main focus is the integration testing.

Unit testing will be used as an additional mechanism of validating the developed code, as a task that each component developer can use in order to verify proper functionality before the integration of the component in ANASTACIA Framework. Usually, all unit-tests are executed during the build-process, unless they are defined to be ignored (marked as @Ignored for Java applications). This practically means that each release of a component has is guaranteed regarding its stability and the same time allows developers to better control the level of test coverage on their components.

The exercise of creating unit tests is still in early stages and the documentation of Unit tests will be provided with more details for all components in deliverable D6.4. An indicative unit test developed in Java code for ANASTACIA is presented in the Listing below.

```
@Test
public void testGetrulesFromInputData () {
```

[11] https://standards.ieee.org/findstds/standard/829-2008.html
[12] https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

```
    List<DataInput> datainputs = dataRepo.findAll();
    for (DataInput datainput: datainputs) {
        List<Attack> rules = ruleRepository.findByExpressionID(datainput);
        if (rules != null) {
            rules.stream().forEach((rule) -> {
                logger.log(Level.INFO, "Rule name {0}", rule.getRuleName());
            });
        }
    }
}//EoM
```

Finally, we introduced a template for reporting the unit tests that will be used for the collecting Unit tests information for deliverable D6.4.

Table 18. Unit Test documentation template

| **Unit Test Case Documentation Form** | |
|---|---|
| Unit Test Reference Code | #UT1 |
| Component | Data Filtering and pre-processing Component |
| Tester | Junit |
| **Short Description** | |
| This test case is responsible for creating, fetching and deleting rules based on incoming monitoring data. | |
| **Input Data** | |
| DataInputType Object | |
| **Output Data** | |
| Success response code | |

Apart from the tests that guarantee the functional correctness of the components, it is important to make tests at integration level for a complete testing and validation process. This means that integration tests shall be created and used for all identified interfaces and to some major platform functionalities. This can be done using unit testing on the methods that are implementing the integration, in order to make them part of a continuous integration and continuous testing process.

## 5.2 Testing for the Integrated Platform

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing in ANASTACIA can also be seen as an extension of unit testing. The main idea of integration testing is to start from two components to test the interface between them. In some cases, more than two components can participate in a common test. Eventually this process will be expanded in order to test all the integrated components of the platform.

The goal of integration testing is to identify problems that occur when components are combined. By using a test plan that suggests the usage of unit tests before combining components, the errors discovered when

in integration tests are most probably related to the interface between them. This method reduces the number of possibilities of errors to a far simpler level of analysis.

In general, integration testing can be done in a variety of ways but the following are three of the most common strategies:

- The top-down approach of integration testing requires the highest-level modules to be tested and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
- The bottom-up approach requires the lowest-level units to be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
- The third approach, sometimes referred to as the umbrella approach, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing.

For the integration testing of ANASTACIA, we chose the last option (umbrella approach), as it combines the best of both approaches. It allows all participating entities, to execute simultaneously multiple testing in several components. In the next section the basic integration tests that have been identified and tested so far, are presented.

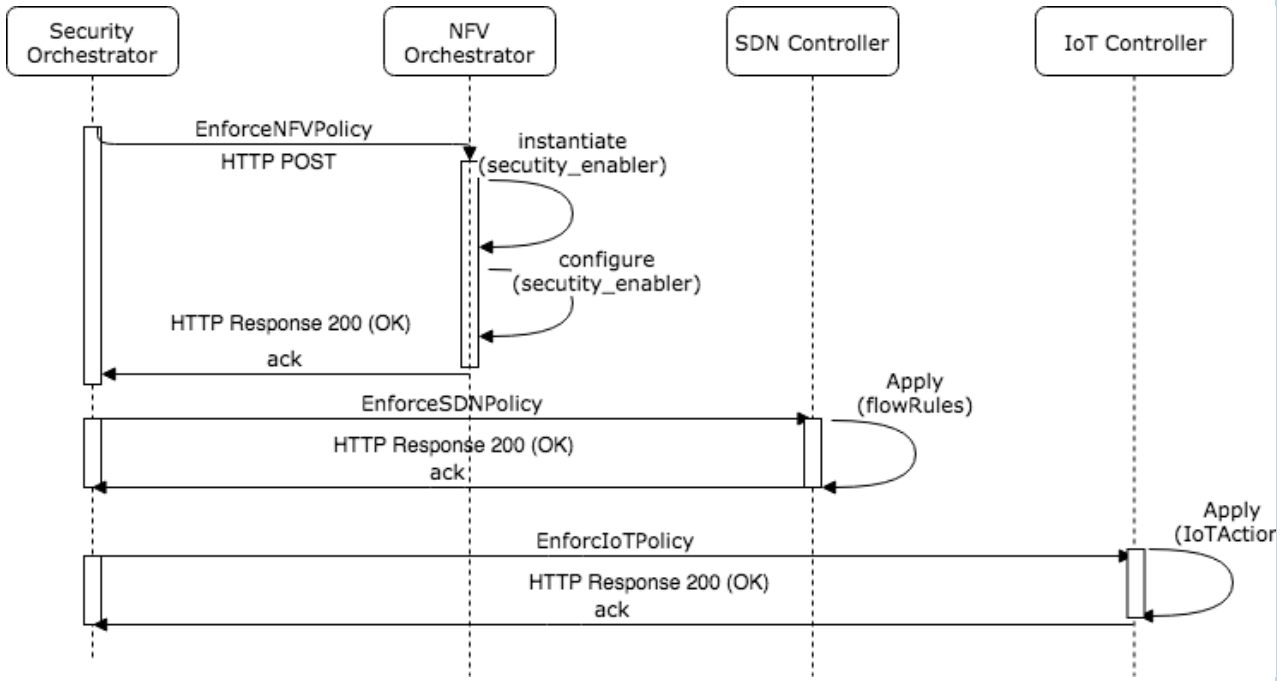
### 5.2.1 Integration Tests

For ensuring the proper integration of the components of ANASTACIA, we will use tests that can assess the functionalities that require multiple components. In these tests, methods from different components are combined in order to achieve the needed functionality. Therefore, the focus is given to the combination of pieces that create a basic integrated functionality. In the following Table 19 the tests that have been identified so far, in order to cover the integration aspects of the project, are presented.

Table 19. Identified and Planned Integration Tests

| **Test** | setAnalysisParamsTest | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| Monitoring Configuration Interface (MCI) | Security orchestrator, Data Filtering and pre-processing Component | Testing of the proper functioning of editing the parameters that affect the data filtering |
| The security orchestrator is responsible for enforcing relevant security policies in the data plane. This can be either using SDN, NFV, the IoT controller or a combination of those. For the NFV part, the security orchestrator will make sure that the security enablers are up and configured by interacting with the NFV orchestrator. When it comes to SDN, using the northbound API, receiving an acknowledgment of the | | |

request means that the security policy has been properly enforced. The same is applicable for the IoT mitigation actions (The reception of the acknowledgement confirms the policy enforcement).




| **Test** | SubscribeData-from-IoT-network | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | IoT broker and Data Filtering and pre-processing Component | Testing of the data exchange of sensors values and attack notifications |

The Data Filtering and pre-processing Component subscribes to the IoT broker to receive new data from sensors. IoT Broker return subscription response. This response is used to exchange new data between IoT broker and Data Filtering and pre-processing Component.

| **Test** | NetworkAuthentication | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | Network Authenticator, IoT device and Orchestrator | Testing the bootstrapping process to authenticate a new device in the IoT network. |

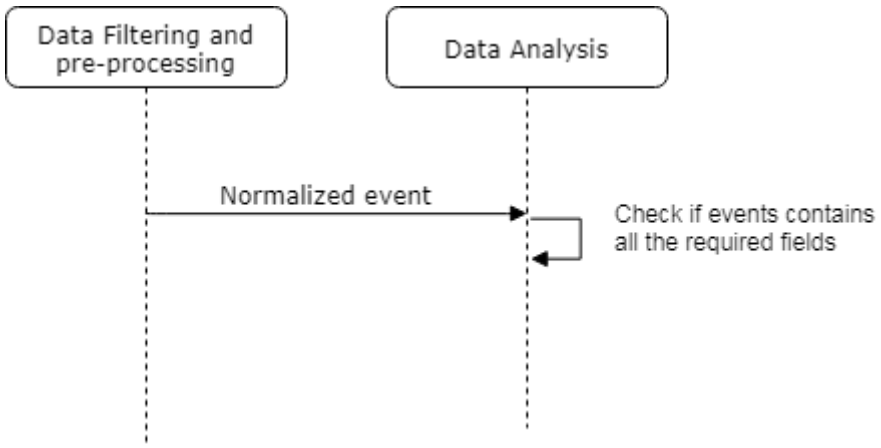
When each IoT device is switched ON, the device sends a message including its identifier to request the authentication in the network. The network authenticator verifies the identifier to enable the communication. If the verification is ok, the network authenticator sends a message including the IP of the new device and the DTLS key generated in the bootstrapping process towards the Orchestrator.

| **Test** | GetCapabilityToken | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |

| | | |
|---|---|---|
| | AAA Architecture, Network Authenticator and IoT device | Testing the query of capability tokens to authorize the data publication in IoT broker |
| The IoT device sends a query including the publication action towards the IoT broker for requesting a capability token. The AAA Architecture verifies the authorization of that device according to XACML policies. If the authorization is ok, the AAA Architecture responds an ACK message including the capability-token towards the IoT device. The Network Authenticator acts as a bridge between IoT device and AAA Architecture. | | |
| **Test** | VerifyCapabilityToken | |
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | IoT device and IoT broker | Testing the verification of capability tokens to authorize the data publication in IoT broker |
| The IoT device sends a message including data and capabilityToken. The IoT broker verifies the capabilityToken before publishing the data in MongoDB. The IoT broker sends a ACK message towards the IoT device. | | |
| **Test** | NetworkAuthentication | |
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | Network Authenticator, IoT device and Orchestrator | Testing the bootstrapping process to authenticate a new device in the IoT network. |
| When each IoT device is switched ON, the device sends a message including its identifier to request the authentication in the network. The network authenticator verifies the identifier to enable the communication. If the verification is ok, the network authenticator sends a message including the IP of the new device and the DTLS key generated in the bootstrapping process towards the Orchestrator. | | |
| **Test** | DTLS-activation | |
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | DTLS proxy and IoT controller | Testing the activation of secure channel using DTLS protocol |
| The IoT controller sends a message including the IP address of IoT device and a shared-key to DTLS proxy which opens a secure channel with the IoT device. | | |
| **Test** | | |
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | Data analysis and Data Filtering pre-processing | Testing that the security event are correctly normalized when |

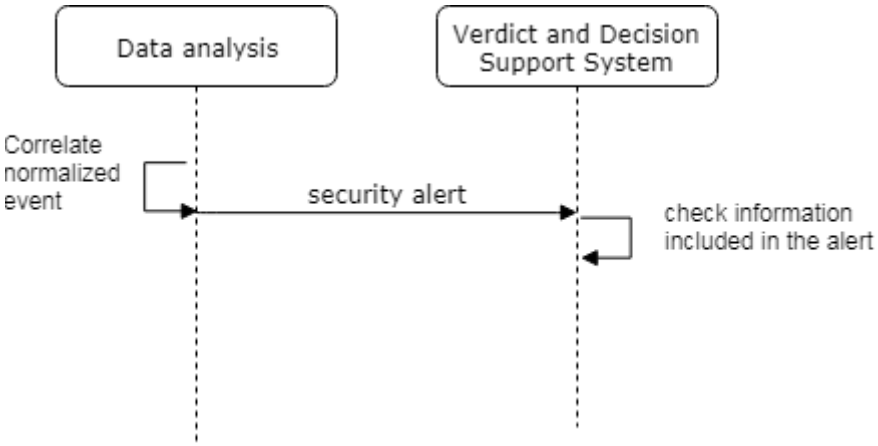
| | | |
|---|---|---|
| | | received from the Data Filtering and pre-processing |

The Data Filtering component sends a probe event to the Data analysis including the basic information: source of attacked devices, detination of the attack, type of attack, timestamp.



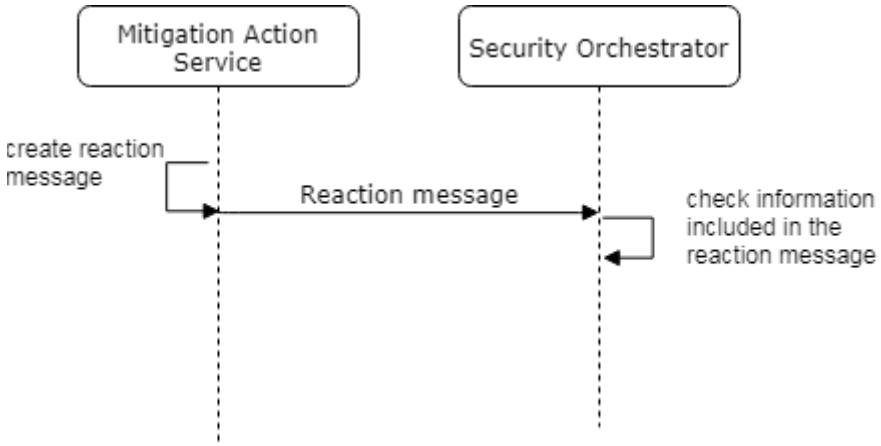

| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| MVI | Data Analysis and Verdict and Decision Support system | Testing that security events are correctly correlated and alarms are generated |

The Data Analysis will correlate events that derive into the generation of alarms. The test wll use a simple rule set at the Decision Support system that will trigger a simple alarm for a single event sent from the Data Analysis.



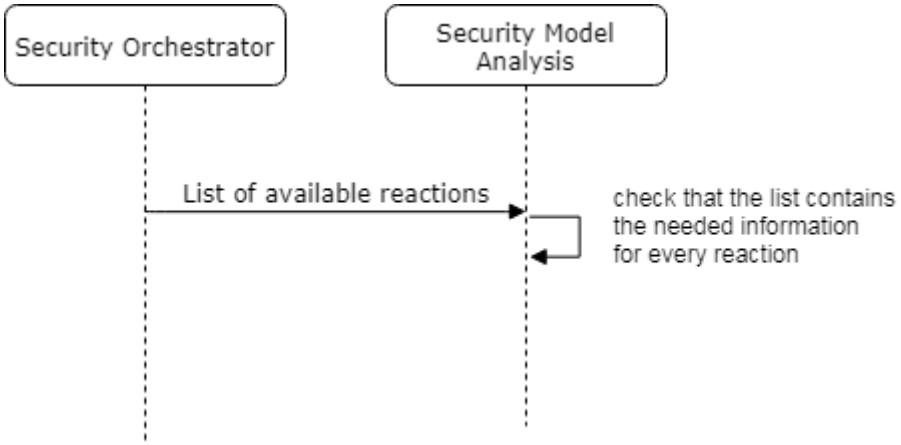

| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| CSI | Mitigation Action Service and Security Orchestrator | Testing that the information required to trigger a reaction is correctly |

The Mitigation Action Service will create a reaction message describing the incident to mitigate and the reaction chosen. The test consist in checking that all the information required to enforce the mitigation is correctly included in the reaction message.



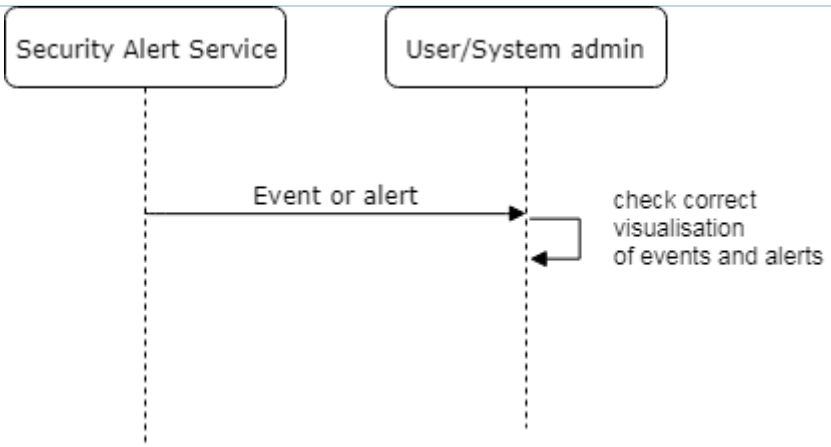

| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| RCI | Security Orchestrator and Security Model Analysis | Testing that the information about reaction available at the platform is correctly received by the Reaction module |

The Security Orchestrator will send a message to the Security Model Analysis listing the reactions that are enforceable by the platform. The test will check the appropriate reception of the information.



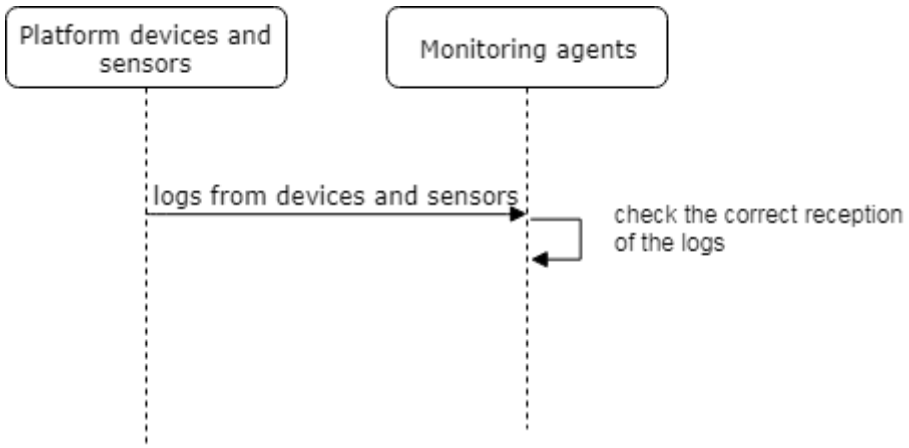

| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| SAWI | Security Alert Service | The test will show a simple event and a simple alert in a dashboard at the system admin side. |

The incidents detected and the event received by the monitoring module will be displays in a GUI. A simple event and simple events will be displayed to check the correct visualization of the events.

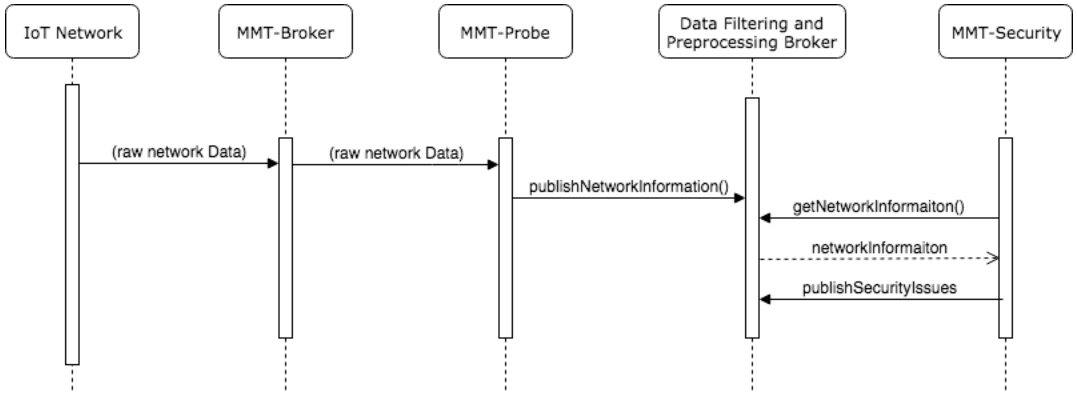



| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | Platform devices and sensors, Monitoring agents | The test will show the correct reception of monitoring data from the platform to the monitoring agents |

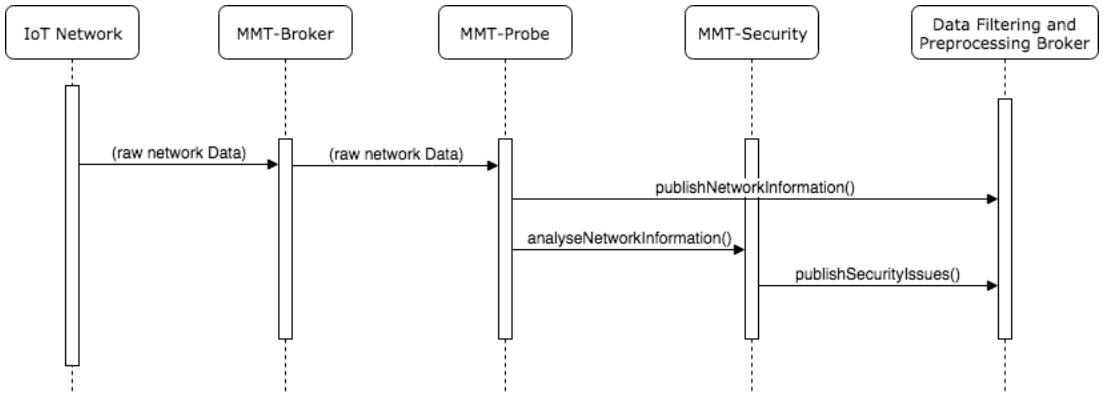
The test will check the correct communication between sensors and devices deployed at the IoT platform.




| **Test** | MMT Integration test (Monitored Data General Interface – MDGI) | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| | IoT Network, **MMT-IoTBroker**, MMT-Probe, MMT Security, Data Filtering and preprocessing Component | Testing of the integration of MMT solution into the ANSTACIA Platform, using the Monitoring Data General Interface |




Approach A




Approach B

NOTE: Two possible approaches are possible here:

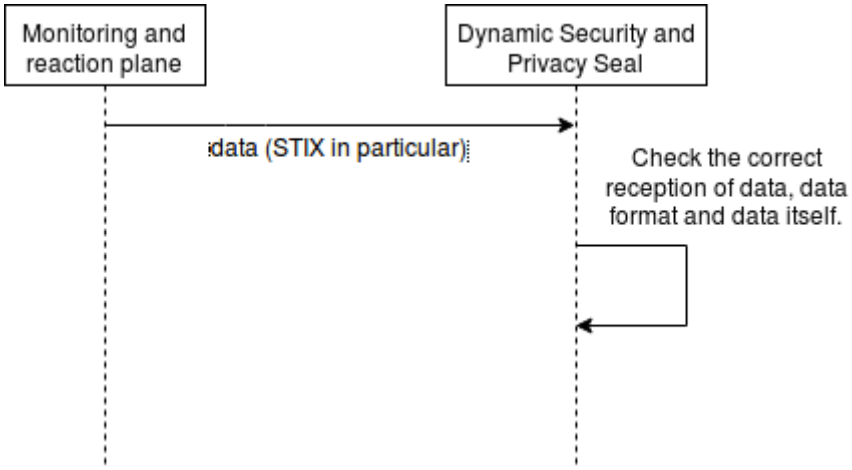
- Approach A: The general ANASTACIA Architecture establishes that the data should be sent from the Agents to the Data Filtering and Pre-processing component, to be sent from this module to the analysis tool. This approach utilizes MMT's capability to use Kafka Brokers to publish the information extracted by MMT-Probe (using DPI techniques), making it available to all analysis tools. In particular, the MMT-Security can use this information from the Kafka Channel in order to test the security properties. The verdicts of these analyses will also be published in the Kafka Broker in order to be used other analysis tools (in particular, the XL SIEM Tool).
- Approach B: In this approach, MMT-Probe both publishes the information on Kafka and sends it directly to MMT-Security. Once the Security analyses have been performed, the verdicts are also published to Kafka.

The first approach has been selected.

| **Test** | | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| SMMI | Dynamic Security and Privacy Seal | The test will display the different kinds of data provided by the modules developed in the frame of WP4. |

The detected threats and the corresponding events transmitted from WP modules to the DSPS server will be shown in the DSPS GUI. The test is separated in several parts, depending on the source of the data,

their format and the information contained. Of course, this test will include the validation of the data described using STIX/TAXII.



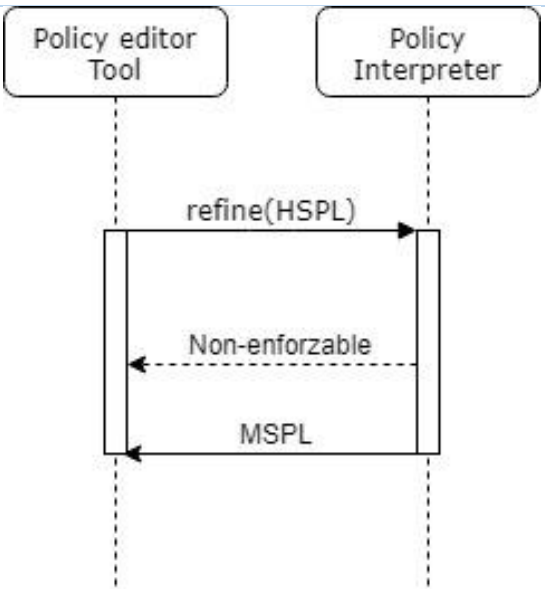

| **Test** | HSPL policy definition | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| - | Policy Editor Tool | Testing the proper HSPL definition from the Policy Editor Tool |

To instantiate HSPL policies through the Policy Editor Tool and verify the file generated is compliant with the HSPL scheme.

| **Test** | HSPL policy refinement | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| H2MI | **Policy Interpreter**, Security Enabler Provider | Testing the proper HSPL refinement |

To perform HSPL policy refinements, verifying:

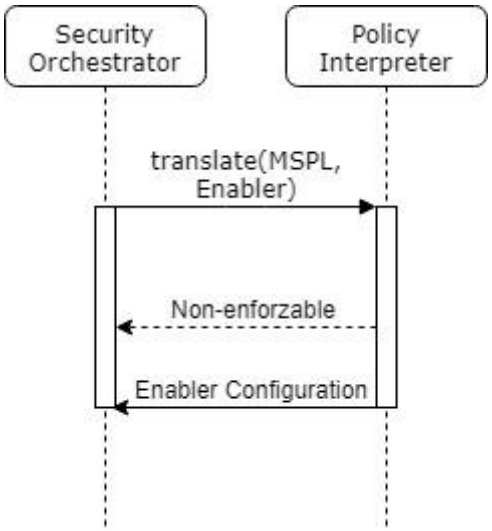
1. Non-enforceable results
2. Enforceable-results, verifying MSPL generated are correct and compliant with the MSPL scheme.




| **Test** | MSPL policy translation | |
|---|---|---|
| **Interface(s) Tested** | **Components Used** | **Short Description** |
| M2LI | **Policy Interpreter**, Security Enabler Provider, Security Orchestrator | Testing the proper MSPL translation into a security enabler configuration. |

To perform MSPL policy translations for the main identified security enablers, verifying:

1. Non-enforceable results
2. Enforceable-results, verifying the generated configuration is correct for the specific technology.




## 5.3 VALIDATION FOR THE INTEGRATED ANASTACIA PLATFORM

For the evaluation of the quality of the developed platform we selected a popular standard as basis, namely the ISO/IEC 25010:2011 “Systems and software engineering - Systems and software Quality Requirements

and Evaluation (SQuaRE) - System and software quality models"[13]. In more detail, the ISO/IEC 25010:2011 defines as stated in its official website:

A quality in use model (in our case Actual Usage Evaluation) composed of five characteristics (some of which are further subdivided into sub-characteristics) that relate to the outcome of interaction when a product is used in a particular context. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.

A product quality model (in our case Technical Evaluation) composed of eight characteristics (which are further subdivided into sub-characteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

Since the technical assessment of ANASTACIA framework will not only be based on the software elements that will be delivered, but also the perceived usefulness and appropriateness will be assessed by the use cases, the technical validation on ANASTACIA can be conducted using a subset of the product quality model of the ISO 25010 and defined Key Performance Indicators (KPIs) based.

### 5.3.1 The Product Quality Model

The product quality model describes the internal and external measures of software quality. Internal measures describe a set of static internal attributes that can be measured. The external measures focus more on software as a black box and describes external attributes that can be measured.

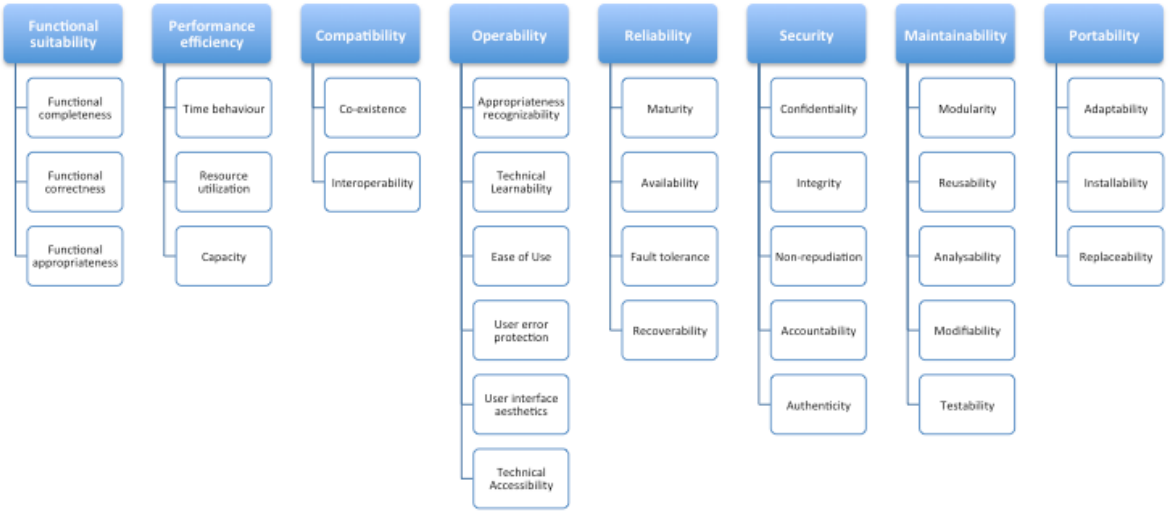



Figure 7. A product quality model view based on the ISO/IEC 25010:2011 standard

In general, this model evaluates software quality using a structured set of characteristics (each of them including other sub-characteristic), which are the following:

1. Functional suitability - The degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions.
2. Performance efficiency - The performance relative to the amount of resources used under stated conditions.
3. Compatibility - The degree to which two or more systems or components can exchange information and/or perform their required functions while sharing the same hardware or software environment.
4. Operability - The degree to which the product has attributes that enable it to be understood, learned, used and attractive to the user, when used under specified conditions.

[13] http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733

5. Reliability - The degree to which a system or component performs specified functions under specified conditions for a specified period of time.
6. Security - The degree of protection of information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.
7. Maintainability - The degree of effectiveness and efficiency with which the product can be modified.
8. Portability - The degree to which a system or component can be effectively and efficiently transferred from one hardware, software or other operational or usage environment to another.

The next table showcases the sub-characteristics of each category and indicates their relativity to the ANASTACIA framework.

Table 20. Technical Characteristics and Sub-characteristics relevant to ANASTACIA technical validation

| Sub-characteristics | Definition | Relation to ANASTACIA integrated platform validation |
|---|---|---|
| **Functional suitability** | | |
| **Functional completeness** | Degree to which the set of functions covers all the specified tasks and user objectives. | YES |
| **Functional correctness** | System provides the correct results with the needed degree of precision. | YES |
| **Functional appropriateness** | The functions facilitate the accomplishment of specified tasks and objectives. | NO |
| **Performance efficiency** | | |
| **Time behaviour** | Response, processing times and throughput rates of a system, when performing its functions, meet requirements. | YES |
| **Resource utilisation** | The amounts and types of resources used by a system, when performing its functions, meet requirements. | YES |
| **Capacity** | The maximum limits of a product or system parameter meet requirements. | NO |
| **Reliability** | | |
| **Maturity** | System meets needs for reliability under normal operation. | YES |

| | | |
|---|---|---|
| **Availability** | System is operational and accessible when required for use. | YES |
| **Fault tolerance** | System operates as intended despite the presence of hardware or software faults. | YES |
| **Recoverability** | System can recover data affected and re-establish the desired state of the system is case of an interruption or a failure. | YES |

#### 5.3.1.1 ANASTACIA framework: Product Quality Evaluation Framework

Based on the tables presented in the previous sections that reveal which criteria could be measured during the ANASTACIA framework operation, and based on the fact that the ISO/IEC 25010:2011 standard does not define specific attributes (measuring ways) for each one of the sub-characteristics, the following list of indicators has been devised in order to allow the technical assessment of the ANASTACIA framework. It needs to be noted that due to the nature of the project and based on the operation conditions of the pilots, some of the below mentioned indicators are considered optional, as their measurement might not be possible/ not produce meaningful results.

Table 21. Quantitative Evaluation Metrics selected for the ANASTACIA framework

| Sub-characteristics | KPIs | Calculation Type |
|---|---|---|
| **Functional completeness** | Portion of functional requirements covered | (Completed high priority functional requirements / Total Number of high priority requirements) * 100 % |
| **Functional correctness** | Portion of functional requirements covered without reported bugs, after tests | (Completed functional requirements of high priority without bugs / Total Number of high priority requirements) * 100 % |
| **Time behaviour** | Average Latency | (Total Response Time)/(No. of Requests) |
| | Throughput | (Total No. of Kilobytes)/(Total Time of Operation) |
| **Resource utilisation** | Mean % CPU Utilisation | (Σ (% CPU utilisation probes))/(No. of probes) |
| | Max. Memory Used | No. of max Megabytes of RAM Memory recorded |
| | Max. Processing Power Used | max % CPU utilisation recorded |
| **Maturity** | Max. Concurrent Users Supported | No. of Max. Concurrent Users Recorded |
| | Simultaneous Requests | No. of Simultaneous Requests |

| | | |
|---|---|---|
| **Availability** | % Monthly Availability | 1- ((Downtown Time Minutes)/(Month Days*24*60)) |
| | Error Rate | (No. of Problematic Requests)/(Total Number of Requests) |
| **Fault tolerance** | Number of Software problems identified without affecting the platform | No. of Non Critical Software Errors |
| **Recoverability** | Mean time to recover from software problems | (Total Recovering Time due to Software Issues)/(Total Software Issues resulting to recovery) |

As ANASTACIA components have very diverse nature, it is not easy to use common measurement and KPIs for all. Therefore, in the following tables Table 22 and Table 23 we try to identify the suitable KPIs for each of these components. A dash means that the metric shall be examined in next stages, while N/A means that the metric is not applicable.

Table 22. KPIs per component (part 1/2)

| | Functional Completeness and Correctness | Average Latency | Throughput | Mean % CPU Utilisation | Max. Memory Used | Max. Processing Power Used |
|---|---|---|---|---|---|---|
| **Policy Editor Tool** | N/A | <1s | N/A | - | - | - |
| **Interpreter** | - | Policy refinement and translation processes for a basic security policy should take < 2s | - | - | - | - |
| **Security Enablers Provider** | - | Depends on the enablers ( e.g. Latency resolve a authorization query should be <10s ) | - | - | Should be minimized, but depends on the overall input load that will be provided | - |
| **Security orchestrator** | 66% | Depends on the desired security policy to be enforced | 1GBpS | Low | 1Gb | 50% |
| **NVF orchestrators** | 60% | Depends on the desired security policy to be enforced | 1GBpS | High | 16Gb | 85% |
| **SDN controllers** | 70% | Depends on the desired security policy to be enforced | 1GBpS | Medium | 8Gb | 70% |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Data Filtering and pre-processing Component** | N/A | Latency should be <1s | Should be high but depends on the installation | Medium | Should be minimized, but depends on the overall input load that will be provided | 85% |
| **Security Alert Service** | Integrity and certain reception of exchanged messages | Latency < 1s | - | Should be minimized. | - | - |
| **Dynamic Security and Privacy Seal component** | Functional, completed, correct | Latency < 1s | - | Should be minimized. | - | - |
| **Dynamic Security and Privacy Seal User Interface** | Functional, completed, correct | The latency should not affect the user but depends on the communication protocol used between the DSPS Web server and the user's terminal (for instance 3G or 4G). | - | Should be minimized. | - | - |
| **ANASTACIA Platform as whole** | **>90% functional completeness and correctness** | **Platform latency <1s in general cases** | **N/A** | **N/A** | **N/A** | **N/A** |

Table 23. KPIs per component (part 2/2)

| | Max. Concurrent Users Supported | Simultaneous Requests | % Monthly Availability | Error Rate | Number of Software problems identified without affecting the platform | Mean time to recover from software problems |
|---|---|---|---|---|---|---|
| **Policy Editor Tool** | N/A | >10 | >99% | Low | - | - |
| **Interpreter** | N/A | >10 | >99% | Low | - | |
| **Security Enablers Provider. AAA Architecture** | N/A | >10 | >99% | Low | - | - |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Security Enablers Provider. Network Authenticator** | N/A | >10 | >99% | Low | - | - |
| **Security Enablers Provider. DTLS proxy** | N/A | >10 | >99% | Low | - | - |
| **Security Enablers Provider** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **Security orchestrator** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **NVF orchestrators** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **SDN controllers** | N/A (Internal Component) | >20 | >99% | Low | 0-multiple (depends on the architecture of the network) | Few minutes |
| **Monitoring Agents** | N/A (Internal Component) | N/A | >99% | Moderate | 0 | Few minutes |
| **Data Filtering and pre-processing Component** | N/A | >300 | >99% | Low | | few hours |
| **Data Analysis Component** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **Security Model Analysis Component** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **Verdict and Decision Support System** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **Mitigation Action Service** | N/A (Internal Component | >20 | >99% | Low | 0 | Few minutes |
| **Security Alert Service** | N/A (Internal Component | >10 | >99% | Very Low | 0 | Few minutes |
| **Dynamic Security and Privacy Seal component** | N/A | >10 | >99% | Low | 0 | Few minutes |
| **Dynamic Security and Privacy Seal User Interface** | More than 20 users on the same instance | >10 | >99% | Low | 0 | Few seconds: an error message can be displayed to the users. |

| | | | | | | |
|---|---|---|---|---|---|---|
| **ANASTACIA Platform as whole** | **>10** | **N/A (depends on the request type)** | **>99** | **Low** | **0** | **Few minutes (all service must start)** |

## 5.4 ANASTACIA DEVELOPMENT LIFECYCLE

From a technical point of view, developing an integrated framework like ANASTACIA through the combination of the developed artefacts is a difficult challenge, especially when trying also to guarantee the quality of the software. For this reason, the identified integration plan is enhanced with the alignment of the way that developers work in the project, through the definition of descriptive steps for the development, technical integration and deployment of components. This proposed development lifecycle scheme for ANASTACIA includes source code management, continuous integration, source-code quality control, release management and ticketing, and is presented in the following section.

### 5.4.1 Source Code Management

For the collaborative and distributed development of software, the usage of a common Version Control System (VCS) and a dedicated code repository is required. A VCS refers to a repository of files, usually for the files of source code of software, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixes, or enhancements introduced, by the change. Modern code repositories and VCSs allow developers to effortlessly work in the same project by providing merging, branching and storing of code functionalities, and the same time provide many functionalities that help collaborative development. Some of the most commonly used Code Repositories are CVS[14], SVN[15] , Mercurial[16] and Git[17].

In ANASTACIA we have selected Git as it is a distributed revision control system (every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server) that is very popular and many online repositories exist in order to accelerate the development work and avoid the overhead of private installation. Git is very fast, provides branching capabilities as core functionality and since most operations are local there is no network latency involved.

Online code repositories as a service, like GitHub, Gitlab or Bitbucket, cover a broad aspect of functionalities, including code repositories with versioning systems and team collaboration tools. Our selection for ANASTACIA is Gitlab, due to ease to create multiple repositories under a single group that is created for ANASTACIA[18], but also due to the offering of container registry, continuous integration and issues management, functionalities that as described in the following sections are part of the ANASTACIA development lifecycle.

### 5.4.2 Continuous Integration

The development and the deployment of ANASTACIA are based on a Continuous Integration (CI) process. CI is a software engineering approach of merging all developer committed copies to a shared registry and test it automatically on each commit or on a regular time (usually every day- nightly builds) to increase software quality since the early stages of development. It is used in software development to automate and improve

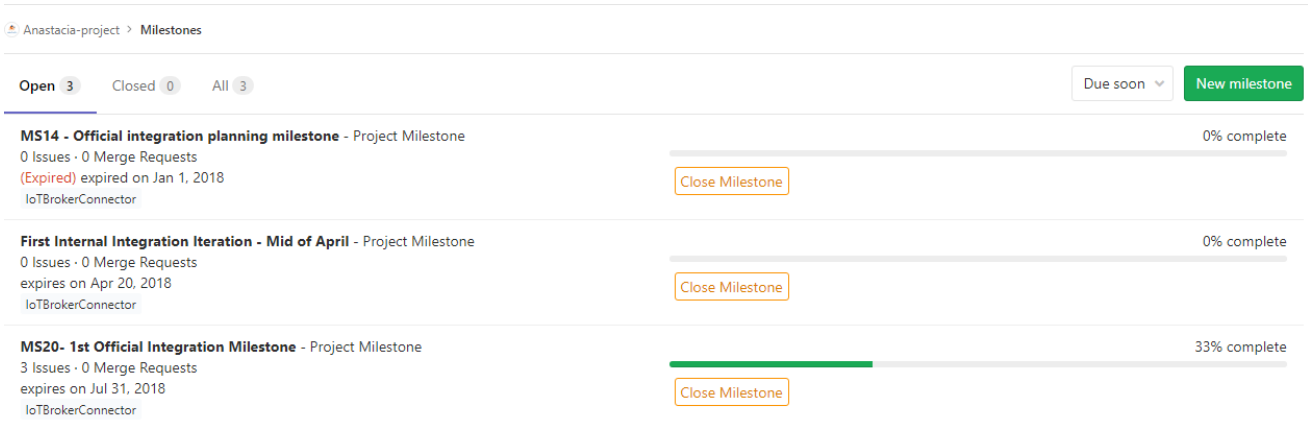
14 http://www.nongnu.org/cvs/
15 https://subversion.apache.org/
16 http://mercurial.selenic.com/
17 http://www.git-scm.com/
18 https://gitlab.com/anastacia-project

the process of software integration. The goal of CI is to ensure that code changes are not breaking the whole solution and allowing rapid and safe deployment of newer versions to production, as it ensures the proper functioning through automated testing. The starting point of this process is usually the commit of code by a developer. For every commit made, a git hook is configured to build in an automated manner new releases of ANASTACIA components and even build the whole framework on a staging environment.

The overall development and integration process in ANASTACIA will be supported by GitLab, through the project collaborative group https://gitlab.com/ANASTACIA-project, as GitLab provides out of the box capabilities for Continuous Build. In order to enable Continuous Integration on a repository, a simple process is needed, consisting of two main steps; a) the addition of a project configuration file in YAML format(.gitlab-ci.yml[19]) in the root folder of the repository b) the configuration of project to use a script that executes called Runner, so that in each commit or push, the CI pipeline will be triggered.

### 5.4.2.1 Release Management

As described in section 3.1, two major releases of the ANASTACIA framework are planned during the project duration. The releases of the framework will be deployed using specific versions of each developed component, using the configuration options that Docker compose file provided. Also, for the better organisation of these releases, we have created in GitLab dedicated Milestones and also other internal milestones for each component. The goal of this approach is to allow ANASTACIA partners to collaborate in order to create the planning for each release in terms of functionalities that should be provided and issues to be resolved.




[19] https://gitlab.com/help/ci/yaml/README.md

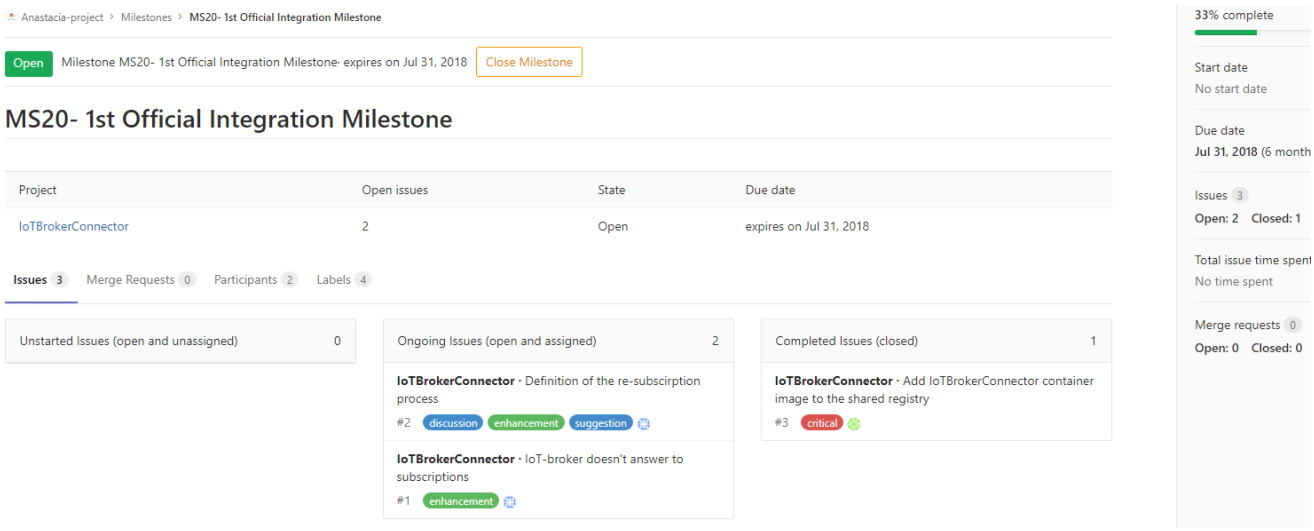



Figure 8. Release Management using GitLab

### 5.4.3 Source Code Quality Control

Another parameter of the validation of the developed software is the quality measurement of source code. Although, quality is somewhat subjective attribute and understood differently by different people, an independent organization, founded by the Software Engineering Institute at Carnegie Mellon University and the Object Management Group, called Consortium for IT Software Quality (CISQ[20]) has defined a set of software structural quality characteristics. In the “House of Quality” model, these are "What’s" that need to be achieved:

- ***Reliability***: An attribute of structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software.
- ***Efficiency***: The source code is the element that ensures high performance once the application is in run-time mode. Efficiency is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount.
- ***Security***: A measure of the probability of potential security breaches due to poor coding practices or architecture. This kind of breaches increases the risk of critical vulnerabilities that can damage a business.
- ***Maintainability***: Maintainability includes the concept of adaptability and portability. It is very important to measure the maintainability for mission-critical applications, where each change is driven by tight schedules and is important to remain responsive during the changes. It is very crucial to keep maintenance costs under control.
- ***Size***: The sizing of source code is a software characteristic that obviously impacts maintainability.

Sonar[21] is an open source software quality platform. Sonar uses various static code analysis tools such as CheckStyle[22], to extract software metrics, which then can be used to improve software quality. The results will be presented in deliverable D6.4 - Final Technical integration and validation Report.

[20] http://it-cisq.org/
[21] http://www.sonarqube.org
[22] http://cruisecontrol.sourceforge.net

In some case the usage of a centralized SonarQube installation might not be possible due legal reasons concerning source code rights. In that case responsible partners have to use their own SW quality tool (ideally SonarQube as quality assurance tool) in order to monitor and commonly agreed KPI´s/Metrics.

#### 5.4.3.1 Defined KPI´s/Metrics

To reach an acceptable level of quality the goal for ANASTACIA project is to reach the following described KPI´s when delivering to the integration environment. A first selection of the metrics and the KPI´s has been done and based on our integration and testing process we will continuously monitor the results to the metrics and adapt the suggested KPI´s if needed. The KPI's are especially important to be respected when a platform release is scheduled. The defined KPIs are the following;

- Number of violations against coding guidelines (critical, high, medium…) per component
  - Critical = 0, when providing a component for integration
  - High = no threshold, but have to be resolved in time
  - Medium: no threshold
  - Code Smells: no threshold
- Technical Dept = A specific target has not defined yet, but should be getting smaller for each iteration we deliver
- Density of comment lines =
  Comment lines / (Lines of code + Comment lines) * 100 : A specific target not defined yet, but we need each function/component to have an header with a description
- Density of duplication = Duplicated lines / Lines * 100 < 10%
- 100% passed unit tests when deliver code for integration
- Condition Coverage = Branch Coverage by unit tests > 40% for new code

### 5.4.4 Issues Management

Collecting in a formal way and managing in the issues that have been raised during the development, integration and testing of the platform is an important step of the developed cycle of any software product. For the issues found during development and technical tests of the platform, the consortium partners are using GitLab embedded issues management approach to store issues, create milestones and distribute the technical work needed to overcome these issues[23]. For the better organization of the issues, specific categories (labels in GitLab) have been created and are used in order to map the issues with the platform components, as depicted in Figure 9.

[23] https://gitlab.com/anastacia-project/framework/issues

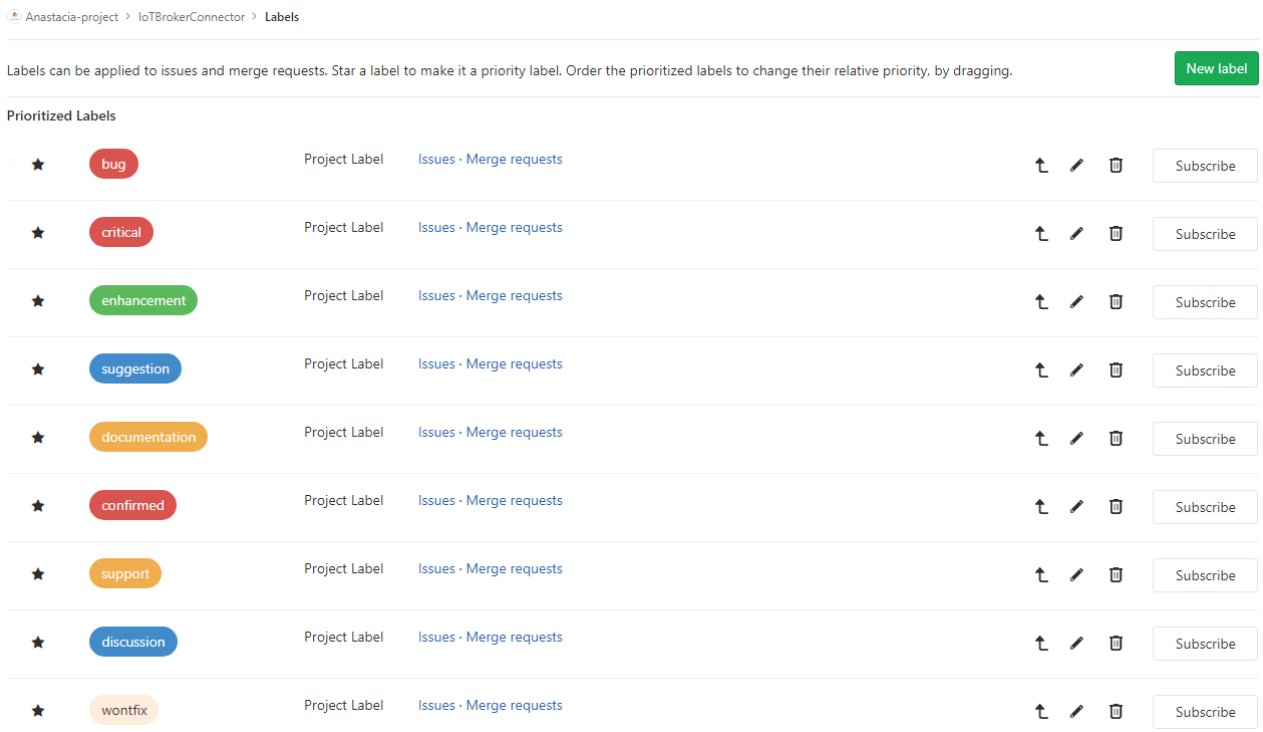



Figure 9. Issues Management for ANASTACIA components - Labels

Finally, in order to impose the time plan for addressing the reported issues, specific Milestones are used as descried in section 5.4.2.1. It is also important to state that the management of issues, labels and milestones is open to all technical partners and is done at repository level, thus changes are made easily when needed. The view of issues however can be aggregated at the root level of ANASTACIA in GitLab, therefore it is easy to have an overview of the issues of whole ANASTACIA framework, as shown in Figure 10.

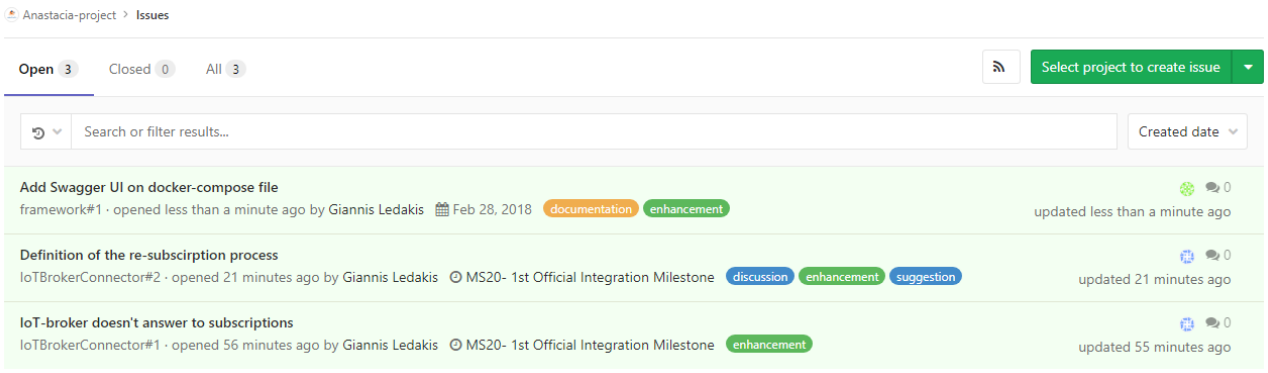



Figure 10. Aggregated issues from all ANASTACIA components

# 6 Conclusions And Next Steps

The current document presented the integration and technical testing plan of the ANASTACIA framework, that is the outcome of the task T.6.1 that has been accomplished with the collaboration all technical partners of the consortium.

Having as starting point for the integration plan both the ANASTACIA Deliverable D1.3 and the work done in task T1.3, we identified the necessary integration points between components and, at the level that this was possible, we concluded on the details of the interfaces that need to be created. These components of ANASTACIA then will be integrated in such a way that they can support common business processes and data sharing across whole framework. For achieving this goal, we researched on different integration methods, and we tried to combine the desired characteristics in order to create an integration approach suitable for ANASTACIA. The integration in ANASTACIA is based on both direct communications between components (Star architecture) and asynchronous, loosely-coupled integration using a common message broker that is scalable by design (Apache Kafka), thus achieving characteristics of event-driven architecture and enable ANASTACIA for the proper supporting of production, detection, consumption of, and reaction to events.

As part of the integration plan, this document provides information regarding the releases of ANASTACIA, with the status of the components and the supported functionalities. It also includes the description of the deployment needs of the various components that will be integrated, so that a better consensus of the whole framework is shaped and proper planning can be done by the use case partners.

For the testing and technical validation of the ANASTACIA platform the “Product Quality Model” from ISO/IEC 25010:2011 has been used as a basis to identify attributes that can be measured and would make sense for ANASTACIA as framework and at component level. Due to the fact that ANASTACIA framework will be deployed and configured at a specific infrastructure for each use case, it is not easy to decide specific KPIs and target values. Nevertheless, an initial effort has been made in order to define this information in the cases this possible and help on the validation of the platform. Also, the definition of integration testing as part of ANASTACIA was provided in this document, along with the integration tests that have been identified so far. Results and also updates of both validation and testing plan will be provided in deliverable D6.4.

Finally, the suggested development cycle and the tools that can be used to support both the collaborative development and the integration of the discrete mechanisms have been provided. The selection of GitLab as basic tool will help the technical to coordinate and collaborate, as it can support our plans for source code management, container registry, continuous integration, and also issue management.

# REFERENCES


[1] Integrated Testing“, http://msdn.microsoft.com/ens/library/12.aspx, 2009

[2] K. Hammer, T. Timmerman, “Fundamentals of Software Integration”, Jones and Bartlet Publishers, 2008

[3] ANASTACIA Grant Agreement N°731558 – Annex I (Part A) – Description of Action, 2016

[4] ANASTACIA Deliverable D1.3 - Initial Architectural Design, 2017

[5] ANASTACIA Deliverable D1.2 – User-centred Requirement Initial Analysis, 2017

[6] N. Josuttis, SOA in Practice. O'Reilly, 2007