A coordinated framework for cyber resilient supply chain systems over complex ICT infrastructures

D4.2 Security and Certification Manager IT1 integration

Document Identification			
Status	Review	**Due Date**	30/09/2021
Version	1.0	**Submission Date**	04/10/2021

Related WP	WP4	**Document Reference**	D4.2
Related Deliverable(s)	D3.1, D3.2, D4.1	**Dissemination Level (*)**	PU
Lead Participant	ATOS	**Lead Author**	Jose Francisco Ruiz (ATOS)
Contributors	POLITO, STS	**Reviewers**	Cataldo Basile, POLITO
			Jan Antič, XLAB

Keywords:
EDC, SACM, SCM, integration, policy, assurance, certification

This document is issued within the frame and for the purpose of the FHISY project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 952644. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Jose Francisco Ruiz	ATOS
José Javier de Vicente	ATOS
Cataldo Basile	POLITO
Grigoris Kaloggianis	STS

Document History			
Version	Date	Change editors	Changes
0.1	05/08/2021	José Javier de Vicente (ATOS)	Table of Contents / work allocation.
0.2	23/08/2021	José Javier de Vicente (ATOS)	First version of section 1
0.3	24/08/2021	José Javier de Vicente (ATOS)	First version of section 4
0.4	06/09/2021	Cataldo Basile (POLITO)	First contribution to section 2
0.5	08/09/2021	Grigoris Kaloggianis (STS)	First contribution to section 3
0.6	20/09/2021	Cataldo Basile (POLITO)	Second contribution to section 2
0.61	21/09/2021	Grigoris Kaloggianis (STS)	Second contribution to section 2
0.7	21/09/2021	José Javier de Vicente (ATOS)	First review and comments
0.8	24/09/2021	José Javier de Vicente (ATOS)	Integration of review and comments. Version for internal review
0.85	30/09/2021	Cataldo Basile (POLITO) and Jan Antič (XLAB)	Internal review
0.9	04/10/2021	José Javier de Vicente (ATOS)	Corrections and improvements after internal review. Version for Quality Check
1.0	04/10/2021	Jose Francisco Ruiz (ATOS), Juan Alonso (ATOS)	Quality assessment and final version to be submitted.

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Jose Francisco Ruiz (ATOS)	01/10/2021
Quality manager	Juan A. Alonso (ATOS)	04/10/2021
Project Coordinator	Jose Francisco Ruiz (ATOS)	04/10/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
D4.2	Deliverable number 2 belonging to WP4
EC	European Commission
EDC	Enforcer & Dynamic Configuration
ICT	Information and Communication Technologies
IRO	Intent-based Resilience Orchestrator & Dashboard
KB	Knowledge Base
KPI	Key Performance Indicator
NFV	Network Function Virtualization
NSF	Network Security Function
SACM	Security Assurance and Certification Manager
SCM	Security and Certification Manager
SeCaM	Security Capability Model
SIA	Security Infrastructure Abstraction
TIM	Trust & Incident Manager
TM	Trust Manager
UI	User Interface
UX	User eXperience
WP	Work Package

Executive Summary

The document describes the implementation plan, approach, and techniques for the Security Certification Manager (SCM) module of the FISHY project. This means the integration of the various components of the SCM module, such as the Enforcer and Dynamic Configuration (EDC) and the Security Assurance Certification Manager (SACM), as well as the SCM that encapsulates EDC and SACM.

The reader will find details on the implementation of the EDC capability model, including the requirements needed for its definition, the workflow proposed for its management and the network security functions considered. EDC integration section also covers the data models considered and the depiction of EDC components.

Regarding SACM, the present deliverable contains a description of the STS tool (which is the basis for the SACM integration) and how the SACM is automated and integrated into FISHY. It is also worth highlighting the addition of a subsection explaining how FISHY is going to monitor the ICT infrastructure.

In addition, the document also addresses SCM. The reader will find an introduction of the use cases and sequence diagrams considered for the development of the SCM architecture as well as the proposed architecture for the component.

The result of the implementation is a first iteration (IT-1) of the platform where the components are just expected to fit and work together and that can serve as a basis to start the process of refinement.

Finally, although the implementation of the various components of the FISHY project requires great coordination, IT-1 or the first prototype represents an important step into the process, acting as a test to determine how effective the integration is, what problems rise from the joining of the different components and what things should be polished and improved for the next prototype. There is little doubt that this work is essential for a satisfactory conclusion and implementation of the FISHY framework.

1 Introduction

1.1 Purpose of the document

This document describes the integration of the Security and Certification Manager (SCM) component into the FISHY architecture. While FISHY WP4 focuses on the development of SCM, this means working and implementing the Enforcement and Dynamic Configuration (EDC) and the Security Assurance and Certification Management (SACM) sub-components.

The integration of the SCM means outcomes of T4.1 (EDC) and T4.2 (SACM) shall properly fit together and, as a result, the whole SCM infrastructure is also prepared to be implemented in FISHY, more specifically in the integration and alignment with WP3 outcomes and WP5 as part of the final integration of the FISHY project.

1.2 Relation to other project work

The SCM component integration considers both the EDC and the SACM integration. These two components have been widely described in FISHY D4.1 [3]. This document provides details on the components' design and implementation for the iteration-1 (IT-1) of the project. Besides, FISHY D5.1 (to be delivered on M15) will assess the overall release and integration of the FISHY platform for IT-1. At the same time, in a similar approach to this D4.2, deliverable D3.2 [2] describes Trust Manager component integration for IT-1, including the assembly of Trust & Incident Manager and Security & Privacy Data Space Infrastructure.

As part of a bigger implementation plan, SCM integration is another key part for the development of the FISHY platform.

1.3 Structure of the document

This document is structured in fiver chapters. The content of each chapter is the following:

Chapter 1 introduces the document, including the objective of the deliverable and its relationship with other documents and work done in the scope of the project, as well as the description of its content and glossary.

Chapter 2 describes the implementation of the Enforcer and Dynamic Configuration (EDC) component. This chapter includes the description of the capability model to be implemented in the FISHY platform, including the requirements for its definition. Chapter 2 also covers the updated information model as well as the description of the Network Security Functions (NSFs) supported.

Chapter 3 presents the integration of the Security Assurance and Certification Manager (or simply SACM). It starts describing the internal components of the Security Assurance Platform. Besides, the chapter includes the description and details about the Evidence Collection Engine and calls needed for the deployment of the component. Finally, the section provides information about the SACM, in terms of monitoring the ICT infrastructure, the steps required for its automation and the APIs required for its integration into the FISHY platform.

Chapter 4 describes the overall SCM integration as part of FISHY task 4.3 leaded by ATOS. This includes the interactions, use cases, sequence diagrams and proposed SCM architecture. In addition, details are provided about each one of the SCM components to be developed and integrated.

Finally, **chapter 5** summarizes the most important conclusions obtained throughout the overall process of integration of the different FISHY components described in the previous chapters.

1.4 Glossary adopted in this document

- **Authentication Header (AH)**: protocol that provides authentication, integrity, and no repudiation. It is included into the IPSec protocol.
- **Encapsulating Security Payload (ESP)**: protocol that manages confidentiality and is part of the IPSec protocol suite.
- **Internet Key Exchange (IKE)**: protocol employed to establish a security association. IKE helps managing secret keys as part of the IPSec protocol.
- **Internet Protocol security (IPSec)**: group of protocols that provide secure communications over IP by means of authenticating and encrypting each packet.
- **Knowledge base**: a central repository for the FISHY project where to store data about threats, attacks… that may be necessary to be available for all components.
- **RabbitMQ**: opensource broker for queueing and delivering messages.
- **Representational State Transfer (REST)**: interface between systems that makes use of HTTP to gather data.
- **XML Schema**: language developed by the W3C and employed to describe structure and restrictions concerning XML documents.

2 EDC integration

As anticipated in D4.1 [3] and confirmed by the update in the milestones, the EDC is in the development phase. However, all the design activities have been performed to ease the lifecycle development and permit the full integration of the EDC in the FISHY architecture in time.

The following sections present the results achieved in the last months in three main areas:

- The design decisions and improvement of the security capability model.
- The security policy models.
- The decisions related to implementing of the EDC components, namely, the Register and Planner, the Enforcer, and the Configurator.

2.1 The capability model (SeCaM)

The development of the capability model is the crucial point in the EDC development. Indeed, almost all the components of the EDC depend on the correct design of this model. Therefore, the improvement of the initial capability model documented in D4.1 [3] has been classified at the highest priority.

2.1.1 Requirements

The following requirements have been identified for the definition of the Security Capability Model:

- The SeCaM describes all the conditions of the NSF that will be considered relevant by the Consortium.
- The SeCaM describes all the actions of the NSF that will be considered relevant by the Consortium.
- The SeCaM describes all the supported resolution strategies supported by the NSF that will be considered relevant by the Consortium.
- The SeCaM introduces all the features that are needed for the specification of policies for (evaluation clauses, default actions).
- The abstract policy language for configuring an NSF is obtained by transformation from an instance of the SeCaM describing it. Equivalently:
    - There must be no need for explicitly defining the abstract language of an NSF if its security capability description is already available.
    - The capability description is the only information needed to define the abstract language of an NSF.
- NSFs having the same security capability description will have the same abstract language.
- All the policies for an NSF will also be valid for all the NSFs that own at least all the security capabilities owned by the first NSF, that is, an abstract policy represented for an NSF will also be valid for all the NSFs that own at least all the security capabilities used in the policy.
- The Register & Planner component requires a root element to store the NSFs available in a specific FISHY domain.
- The instructions for translating the abstract policies of an NSF in configuration settings using its proprietary format must be represented in the security capability model, that is:
    - A translator will need no additional code to support a new NSF, and
    - The security capability model makes is possible to describe in the information model how the translator will generate the actual low-level configuration.

2.1.2 The information model

The information model presented in D4.1 [3] has been slightly updated (see Figure 1 - The capability information model) due to recent research findings and a better characterization of the requirements:

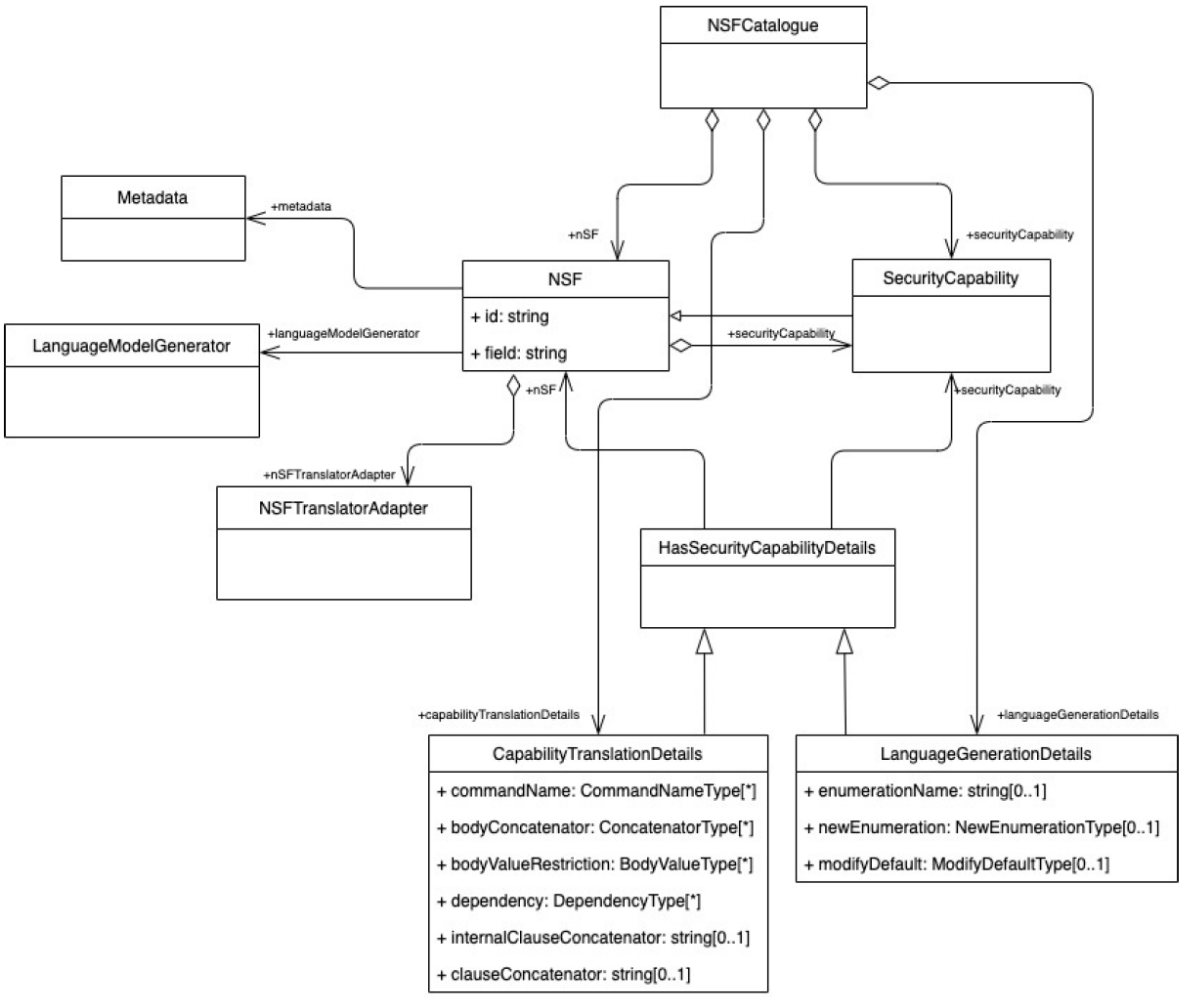

Figure 1 - The capability information model

1. The model has been provided with the NSFCatalogue class, the central aggregator of NSF instances, and the root element of the capability part of the repository.
2. Even if it is an association class, conceptually, the *HasSecurityCapabilityDetail* entity has been expanded into a (normal) class associated with two separate associations to the NSF they characterize:
   - HasSecurityCapabilityDetail.nsf and
   - HasSecurityCapabilityDetail.securityCapability associations).

   Indeed, the association classes are certainly elegant at the modelling level. However, they are not supported in almost all the implementation level formats and tools (e.g., XMLSchema, databases).
3. The subclasses of *HasSecurityCapabilityDetail* have been organized based on the functions that will use the information they store. Moreover, these classes have been renamed: instead of maintaining the names of the design patterns used to find the solution, we have preferred more self-explanatory names. Therefore, the two subclasses are now:

- CapabilityTranslationDetails, reports information the translator will use to generate the low-level configurations.
- LanguageGeneratorDetails reports information for the language generator to be used when generating the abstract language associated with the NSF.

2.1.3 Capability model: management workflow and formats

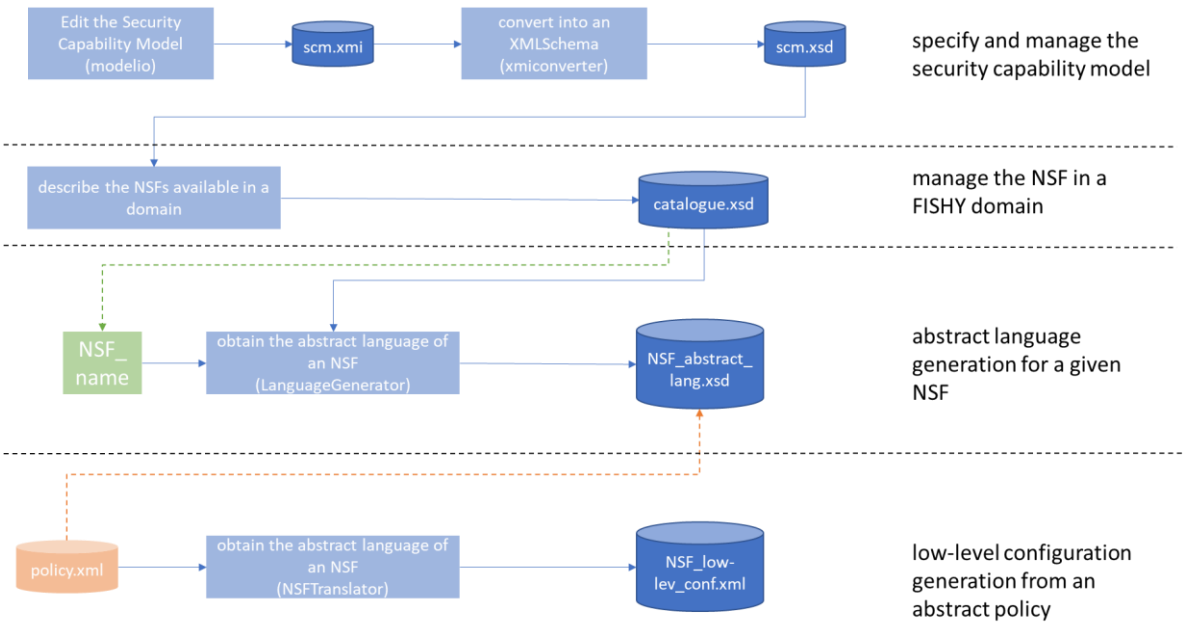

Figure 2 - Security Capability Model: Management workflow

Easing the management of the activities related to the development and maintenance of the security capability model is a main objective for the FISHY project. This activity required a careful workflow design. Accordingly, the following decisions have been made during these months:

The Security Capability model is maintained as a UML class diagram.

The UML class diagram includes both the information model, the data models needed to represent categories of NSFs (e.g., filtering devices, VPN gateways). Moreover, it includes all the data types needed to characterize the class instance values and the attribute types.

Modelio is the open-source tool used to represent it.

According to our internal analysis, the decision has been made because Modelio is probably the most powerful tool that is both open source and free of charge. In case additional features are needed (e.g., integration with versioning systems), a commercial version is available, providing the most advanced features owned by more expensive suites.

The UML model is exported into an XMI representation.

Representing UML class diagrams with XMI is the de facto standard. However, several dialects and variants make the tools incompatible, almost all due to bad practices from commercial tools. The tests we performed have proved that the XMI format exported by Modelio can be passed with minor issues to more complete tools on the market (e.g., from Altova, Sparks, Visual Paradigm). Proper scripting can recover these issues (e.g., some associations are not recognized, propagation of abstract information not properly managed).

We have decided that the practical format to represent instances will be XML. The class diagram will be translated into an XMLSchema to allow validation.

While it is not the most modern approach as more compact representations are usually preferred nowadays (e.g., JSON), the XML ecosystem is well recognized, well supported with tools, libraries for all the programming languages, validators, parsers, and everything one may need when writing programs that use XML data. Last but not least, XML databases exist and are optimized to store XML objects efficiently, which seems the best way to implement the Register and Planner.

The translation from the security capability model XMI file and the XMLSchema is performed using an ad hoc tool.

The tool, named XSDConverter, takes as input the XMI and produces as output the XSD containing all the classes in the UML model and the defined types. It is written in Java. The sintax for the translation would be the following:

```
java –jar xmiconverter.jar input_filename.xmi [output_filename.xsd]
```

The Catalogue is represented as an XML file and is validated against the XMLSchema.

The list of NSF available in a domain will be maintained in a local catalogue and accessed through the Register&Planner. A catalog is an instance of the NSFCatalogue class. Its correctness will be validated against the XMLSchema.

The generation of the abstract language of an input NSF is generated by an ad hoc tool named LanguageModelGenerator.

```
java –jar LanguageGenerator.jar catalogue_filename.xml \\
                                 nsfName \\
                                 [output_filename.xsd]
```

The LanguageModelGenerator takes as input the XMLSchema describing the Catalogue, the XML describing the security capability model, and the name of an NSF in the Catalogue. This tool exports an XMLSchema that defines the syntax of the policies of the NSF can enforce, i.e., its abstract language specification.

All the capabilities owned by an NSF are known, as they have been specified in the Catalogue, and their characterization is available in the security capability model.

Therefore, the abstract language allows selecting (i.e., the XMLSchema allows specifying) only conditions, actions, events, resolutions strategies, and condition clause evaluation functions that are supported. Thanks to the expressiveness of the security capability model, the abstract configuration language syntax also captures the information about how NSFs support the default actions.

The abstract configuration for an NSF is validated against its XMLSchema using a standard XML validator.

If a policy editor writes a policy for an NSF, he can immediately check the validity against the XML Schema associated with the NSF. Since it is a standard XML validation, XML libraries natively implement this feature.

A policy for an NSF written in its abstract configuration language is translated by means of an ad hoc tool named NSFTranslator.

The NSFTranslator takes as input the XMLSchema describing the Catalogue, the XML describing the security capability model, and the name of an NSF in the Catalogue, and the XML describing a valid policy for the NSF. It outputs the low-level configuration for the input NSF. The low-level

configuration contains the configuration file for the NSF and some FISHY-related management information.

The NSFTranslator reads and understands the *CapabilityTranslationDetails* instances that describe how to translate capabilities written in the abstract language for a target NSF.

```
java –jar NSFTranslator.jar catalogue_filename.xml
                            policy.xml
                            [output_filename.xsd]
```

For the NSFTranslator we are considering several additional optional parameters that allow writing fixed leading or trailing strings for each rule and for the whole policy. The final set of parameters is currently under development and will be presented in the next deliverable, which is D4.3.

2.1.4 Supported NSFs

Currently, the data models allow describing the first categories of security controls:

- Controls able to filter traffic up to layer4, and
- Channel protection controls, like devices for the creation IPsec-based VPN, which apply use cryptography to apply integrity, data authentication, and confidentiality properties to data transferred between two network entities.

The list now includes:

- iptables (https://netfilter.org/news.html), a very famous and effective filtering control available in the Linux distributions (only the filtering modules, no NAT/NAPT)
- XRFM (https://man7.org/linux/man-pages/man8/ip-xfrm.8.html), the native IPsec configuration for Linux platforms
- strongswan (https://www.strongswan.org/), one of the most spread IPsec and IKE implementations.

Moreover, the model supports the following generic security controls (i.e., they are abstractions of security control that do not correspond to existing products; thus, the translation is not needed):

- Generic packet filter implementing the 5-tuple paradigm (i.e., Allow and Deny based on conditions on IP source and destination addresses, IP Protocol Type, source, and destination ports).
- Packet filter with full TCP stateless and stateful filtering
- Generic IPsec-based channel protection module, which supports both Authentication Header (AH) and Encapsulation Security Payload (ESP) with both tunnel and transport mode.
- Generic IKEv1 module.
- Generic IKEv2 module.

These generic NSFs serve as templates and are useful to define standard features to compare the NSFs with.

Supporting these security controls required the modelling of several entities, which have been listed below:

- (*actions*) two filtering actions (allow/deny) and some variants (reject-with ICMP, allow-but-log);
- (*actions*) a data model for describing IPsec actions and features
    - AH vs. ESP, and the general model to describe the parts of the packets (payload or header) covered by the protection.
    - Transport vs. tunnel mode, and the general model to describe different forms of encapsulation that have been introduced in several IETF RFC.

    - The cryptography information needed to create IPsec Security Associations (e.g., MAC algorithms, symmetric encryption algorithms and modes, authenticated encryption).
- (*actions*) an integration of the IPsec actions and features data model to describe IKE-based key-agreement capabilities (DH, RSA, manual specification of security associations).
- (*conditions*) conditions on all the parameters for the most used protocols headers, ranging from layer2 (e.g., MAC addressed) up to layer4 (IP, TCP, UDP) header fields and some session-level protocols (e.g., TLS handshake)
    - Exact match conditions, based on predefined enumerates (e.g., the IP Protocol Type Condition), where an order cannot be specified.
    - Conditions on integer values (e.g., TCP ports) as well as the possibility to specify single values, ranges, and lists.
- (*conditions*) filters to specify stateful conditions on specific protocols (e.g., on the TCP three-way handshake) and networking data (e.g., max bandwidth, max number of connections).
- (*conditions*) a preliminary definition of string match conditions and regular expressions.
- (*resolution strategies*) the First Matching Rule resolution strategies and its variants (e.g., rule chains and Last Matching Rule).
- (condition clause evaluation) DNF and CNF logical formulas.

The model has been validated against the first existing security controls. The validation of the model for a given NSF included the test for:

- The possibility to specify all the features owned by the NSF (i.e., its capabilities).
- The possibility to express a valid configuration written in the NSF-specific configuration language using its abstract language (automatically generated from its capabilities).
- The possibility to translate a policy for an NSF written using its abstract language into a valid configuration written in the NSF-specific configuration language (accepted as valid by the NSF, semantics manually validated by experts).

2.1.5 Availability

All the data models related to the security capability model are available at the project partners here (one folder for each data model).

The Consortium is discussing if they are ready to be made public or if could be necessary to wait for more mature versions.

2.2 Data models

The following sections report the updates for all the other data models that are currently under definition in the FISHY project.

2.2.1 High-level policy language (HLP)

The HLP further progressed during the last month. HLP purpose basic characteristics have been anticipated in the deliverable in D4.1 [3]. The high-level policies (HLP) are security requirements expressed in a way that is independent of technology, security controls and enforcing NSFs, and agnostic to the actual landscape where they will be deployed. The current FISHY HLP is based on the High-Level Security Policy Language (HSPL) defined in the SECURED project.

2.2.1.1 Requirements

The definition of HSPL must satisfy the following requirements:

- Abstraction: HLP should be abstract enough to allow a proper representation of threats and map the result of the compilation of the intents.
- Expressiveness: HLP should be able to support the specification of every type of security policy, also supporting specific conditions (e.g., time constraints, content types, traffic types), for every security application; the HLP must be able to support the specification of all the security policies and constraints that are needed for the FISHY use cases.
- Extensibility: HLP must support future extensions, e.g., by introducing new policy types and specific conditions without the need for changing the structure of language.
- Compatibility: HLP must be compatible with the intents and the intent compiler defined in FISHY WP5.
- Refinement: HLP will be defined so that the refinement of HLP statements into policies written in the abstract language is possible.

2.2.1.2 HLP Definition

HPL follows the typical structure of several authorization languages:

[subject] action object [(field_type,value) ... (field_type,value)]

where:

- ***subject*** (optional) is the user who needs to access or perform some operation on an object (e.g., employee, family member) and may be omitted if the policy is applied to the user that defines the HPL.
- ***action*** is the operation performed on the object.
- ***object*** is the entity (i.e., a resource such as an email scanning, Internet traffic, P2P traffic) target of the action (e.g., authorize access).
- ***(field_type,value)*** is an optional condition that adds specific constraints to the action (e.g., time, content type, traffic type). The value part is a string with a specific format depending on the field type.

The HLP will be defined using XML. Accordingly, the definition of the HLP is made using:

- The main XMLSchema (*hspl.sxd*).
- Four separate XML schemas, one for each of the four fields of HLP, included through standard XMLSchema methods (include schema location).

The schema *actions.xsd* includes (as a data type based on an enumerator) all the actions that are allowed on the object resources. Waiting for the use cases to be completely specified with greater details, this schema allows the specification of:

- Standard authorization actions (e.g., permit and deny access, redirect) for filtering internet connections.
- Actions to require channel protection (e.g., protect).
- Actions to regulate the intensity of elements (e.g., reduce, limit, increase).
- Generic actions to load ad hoc security controls and checks (e.g., enable).

The schema *objects.xsd* includes (as a data type based on an enumerate) all the objects that are the target of HLP statements. Waiting for the use cases to be specified entirely with more significant details, this schema allows reference to:

- Generic objects (e.g., service, or services).
- Standard networking information (e.g., the Internet, subnets, connections, sessions).

- Traffic information (e.g., VoIP, 5G traffic, web traffic).
- Security controls (e.g., logging, antivirus, email scanning, parental control).

The schema *fields.xsd* includes all the constraints that it is possible to specify to characterize the application of the HLP statements. They usually restrict the target defined by the objects. Waiting for the use cases to be completely specified with greater details, this schema allows to select based on:

- Time information, allows fine-grained specification, e.g., generic time of the day (from 10:00 to 11:00), day of the week (e.g., Monday), week of the year (e.g., week 10), months (e.g., august), and generic time (e.g., lunchtime, night, end-of-business).
- Traffic targets, allows the possibility to specify individual machine names (e.g., DNS names), labelled network entities (e.g., the administration subnet), or refer to the traffic generated by the specific subjects (e.g., user1's traffic).
- L3-L7 filters, allows specifying IP addresses (discouraged manually, may be useful when compiling intents), protocol header information (e.g., TCP ports), and URLs.
- Type of content allows referring to a set of predefined categories of the traffic or of the data transferred on the network (e.g., social networks, gambling, illegal websites).

In the file *fields.xsd*, all the field restrictions have been defined by ad hoc data types; some are generic numeric types (e.g., ports), some custom regex (e.g., IP addresses), and some simple enumerator types (e.g., the type of contents).

The schema *subjects.xsd* has not yet been characterized. It only includes generic definitions (e.g., user, administrator, FISHY user, employee) that will be refined once the scenarios are completed.

The final element for the definition of the HLP is a matrix, maintained in the file *constraints.XML* which defines all the valid combinations of actions, objects, and fields. For instance, this matrix states that is possible to specify the following policy (i.e., the combination of values from the four categories):

[all users] 'is not authorized to access' 'Internet traffic' (type of content, {illegal websites})

This matrix needs to be maintained up to date. As soon as new elements are inserted in the types specified previously, the matrix must report the allowed combinations.

This file is read and interpreted by the HLPValidator, a Java object that is used to check if the HLP statements are correct, which is invoked using the following syntax:

HLPValidator.jar hlp_policy.xml

The HLPValidator also performs the standard validation of the XML file against the XMLSchemas listed above.

2.2.2 Medium-level policy languages (aka abstract languages)

Given the current development of the Security Capability Model, we confirm that there will be no need for an explicit definition of a medium-level policy language or languages. The abstract languages will be generated ad hoc for each NSF starting from the description of its capability, i.e., an instance of the Security Capability Model that describes all the capabilities the NSF owns.

2.2.3 Low-level configuration settings (aka configurations)

The design of the additional information used to characterize strongly relies on the features exposed and information required by the SIA. Therefore, this activity has not made significant progress since the last deliverable, D4.1 [3]. This activity has been delayed until the design and implementation of SIA are stable.

2.2.4 Landscape model

There are no significant updates on the models to describe the landscape scenario (i.e., the networked architecture of the domain where FISHY is being used) as the SIA design has not been completed yet. Moreover, there have been no major updates on the formal models used by NFV orchestrators since the last deliverable. Therefore, the ongoing analysis did not need to be updated.

As confirmed by the Consortium, this model is at a lower priority than the security capability model and the high and medium-level policy models. We expect to have more precise requirements and case studies as soon as they are specified more precisely.

2.3 EDC Components status update

2.3.1 Register and Planner

The Register and Planner can be considered a front end to the instances of the NSFs in the Catalogue, represented according to the capability model. In this case, the dependency from the capability model is very evident.

We have decided that the Register and Planner will implement at least the following functions:

- Search functions, e.g., list the NSFs that own a specific capability.
- Comparisons, e.g., answering questions like “are nsf1 and nsf2 equivalent in terms of security policy enforcement?”.
- Enforcement, e.g., answering questions like “is nsf1 able to enforce this policy expressed using the abstract language?”

Some decisions, agreements, and progress include the following ones:

- We have decided that this component will be implemented as a web service.
- The analysis performed has confirmed that using an XML Database is the most convenient solution.
- Currently, we are testing BaseX, which is the open-source (free of charge) alternative having the largest number of features. Currently, eXist is the alternative in case BaseX fails in satisfying all our requirements.

2.3.2 Controller

To perform its refinement tasks, the Controller needs to perform the following basic operations on its two inputs:

- Understanding the security requirements expressed in HLP.
- Understanding the functions that the NSF can provide, expressed using the Security Capability Model.

The difficult part of the refinement process is assigning semantics to all the known concepts in HLP and the features that NSFs offer to implement them. For this purpose, a forward reasoning engine will be used.

Moreover, the Controller will output the refined policies using the abstract configuration language (Medium-Level Policy Language), derived from instances of NSF capabilities. However, the Security Capability Model is under development, and the HLP is stable, but more concrete cases are needed from the use cases.

Therefore, the Controller design, as expected, will start once the formats are stable. Nonetheless:

- Initial effort has been made during these months to determine the refinement process.
- Initial effort has been made to determine the remediation strategies, which will be fine-tuned based on standard scenarios inspired by the FISHY use cases.

At the implementation level, all the decisions made in D4.1 [3] have been confirmed. Moreover, we decided that Enforcer will be provided as a web service. Its architecture will be modular (vertical cuts based on the HLP policies) so that all the components will be implemented in the best possible way.

Currently, several parts of the T4.1 workflow are implemented in Java. Nonetheless, we are investigating the use of Python to ease the development of all the components and modules that are not dependent on Java. For instance, it is unclear if the DRools forward reasoning, the engine we plan to use to automate reasoning about policies concepts, can be called a Python API.

2.3.3 Enforcer

The Enforcer is composed of two main modules:

- One module is in charge of translating abstract policies into low-level configurations.
- The other will interact with SIA to enforce the policies (changes in the network layout, deploy configurations).

The current Security Capability Model design has superseded the need for a translator, which provides translation abilities natively. However, this module will be provided as a web service instead of resorting to a local jar file.

The design module that will interact with SIA has been delayed. It depends on the SIA API and does not appear to pose challenging research issues.

2.4 Risks

Currently, the integration risks that have been highlighted in T4.1 do not appear to create major issues:

- The low-level configuration language parameters depend on the actual features SIA will provide. A delay on SIA has a minimal impact on the language definition. Mitigation does not appear needed at the moment. The risk is considered LOW.
- The Configurator module in charge of deploying policies depends on the SIA API. A delay in defining the SIA features and corresponding APIs has a low impact and does not require mitigation (LOW risk). A reduction of the features exposed by SIA can limit the effectiveness of policy deployment and has significant consequences (MEDIUM risk). Interacting with WP5 and continuing the update of the requirements is mitigating this risk.
- The actual HLP definition and expressiveness depend on the use cases. While Use Cases are under development, this activity requires a very fine-grained definition of all the security requirements. A delay in the formal definition of the use cases may impact the progress in several activities related to the refinement and performed by the Enforcer. The risk that the HLP cannot be adequately specified is LOW, being based on a well-known authorization paradigm (LOW risk). However, the refinement process may not be able to support all the policy types that may be needed for securing the use cases. This risk is considered MEDIUM.

3 SACM integration

The STS Security Assurance Platform (SAP) is an integrated framework of models, processes, and tools to enable the certification of security properties of services. It uses different types of evidence to demonstrate the support for the required properties and award the corresponding certificate.

The following sections present a detailed description of the STS Security Assurance Platform while relative information regarding the integration process to the FISHY platform will be described. Extending the existing Security Assurance solutions offered by STS, the SACM tool will, through an Evidence Collection Engine developed for the purpose, monitor critical components and processes of the ICT infrastructure (leveraging monitoring mechanisms developed in the context of Task 5.1). Based on that input, the tool will provide an evidence-based, certifiable view of the security posture of the ICT system, with accountability provisions for changes that occur in said posture and the analysis of their cascading effects, supporting the runtime checking based on sets of associated claims and metrics.

The mechanisms developed within this task will also enable and provide the design of audit procedures in ICT systems by considering all ICT components within the supply chain. Finally, the methodology and procedures for the automation of security certification are also part of this task, providing different certification models tailored to, e.g., specific security standards, service level agreements, or legal and regulatory obligations (e.g., GDPR).

3.1 STS Security Assurance solution

The Security Assurance Platform is comprised of five basic software modules:

1. **Asset Loader Module**: The component responsible for receiving the cyber system's asset model for the target organization. This model includes the assets of the organisation, security properties for these assets, threats that may violate these properties, and the security controls that protect the assets and is based on STS's Assurance Model. The latter data are defined by the target organization using an excel file provided by STS. This excel file is parsed by the Asset Loader Module, which automatically constructs the respective model for the target organization.
2. **Vulnerabilities Loader Module**: The component responsible for loading the known vulnerabilities of the identified assets and updating the assurance platform depending on the organization's assets included in the assurance model. It is composed of two subcomponents, (a) the Vulnerabilities Loader and (b) the Vulnerabilities Database. Vulnerabilities Loader Module uses OpenVas[1], an open-source vulnerability assessment framework/scanner, while the known vulnerabilities are loaded through a daily update feed provided by Greenbone[2].
3. **Monitoring - Auditing Module**: This component is a runtime monitoring engine built in Java that offers an API for establishing the monitoring rules to be checked. This module is composed of two submodules: (a) the *monitoring database* and (b) the *monitor*. The role of the module is to forward the runtime events from the application's monitored properties and finally obtain the monitoring results. The latter are stored to the monitor database while the monitor submodule. Monitor is the core submodule of the auditing module that reasons if a monitoring rule is violated or satisfied.

[1] https://www.openvas.org/
[2] https://www.greenbone.net/en/security-feed/

4. **Evidence collection - Event Captor Module**: The Event Captor is a tool that, based on collected data and triggering events, formulates a rule or a set of rules and pushes the latter towards the monitoring module for evaluation. Data and events are mostly collected through Elastisearch[3] based on lightweight shippers (namely Beats), such as Filebeat[4], MetricBeat[5], and PacketBeat, which centralizes log data. Data can also be collected through Logstash[6], an open server-side data processing pipeline that ingests data from a multitude of sources, which transforms and then sends them to ElasticSearch. The Event Captor is initiated through the respective REST calls from the monitoring module.
5. **Dynamic Testing Module**: The component responsible for initiating the testing assessment. The module consists of two components: (a) the *dynamic tester* or *manager* and (b) the *dynamic testing tool*.

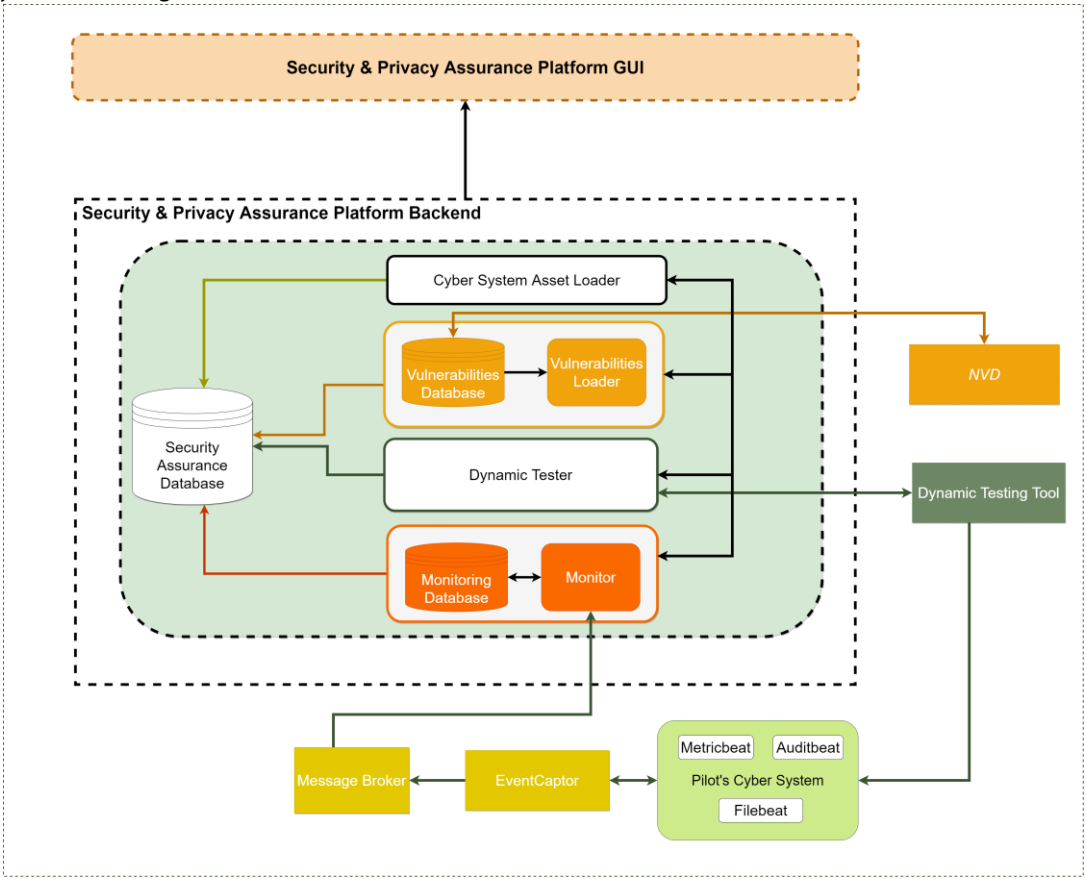

Figure 3 - The Security Assurance Platform Architecture

The STS Security Assurance platform provides the following functionalities:

- Combines runtime monitoring and dynamic runtime testing to ensure the correct and effective operation of security controls.
- Can be hooked to different systems programmatically through appropriate probes (e.g., event captors, test tools) to obtain the monitoring and/or test evidence required for assurance and/or certification assessments.

[3] https://www.elastic.co/
[4] https://www.elastic.co/beats/filebeat
[5] https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html
[6] https://www.elastic.co/logstash/

- Operates based on models that determine the operational evidence that should be captured from systems and how it should be assessed (e.g., what conditions it should satisfy) to assess the correctness and effectiveness of implemented system security controls.
- Enables the runtime assessment of temporal event patterns and rules that can express signature or anomaly-based patterns.

3.2 Evidence Collection Engine

3.2.1 Description of the Evidence Collection Engine

The Evidence Collection Engine or Event Captor Module will be used for real-time, continuous assessment of the security posture of the FISHY's platform while it will aggregate, in real-time, cross-layer evidence pertinent to the security posture of each monitored component. This module will use incoming data from event captors. These are a set of software components that formulate a rule, or a set of rules based on collected data and triggering events and push them towards the Monitoring Module for evaluation. The event captor's module is integrated as a simple dockerized agent/container in the systems that need to be assessed. The purpose of the agents is to gather evidence from various sources (e.g., Network traffic, Security logs, System logs, etc.) and wrap them in an event format, understood by the monitor. The events are then forwarded to the monitor through the event collector.

The Event Captor Module is developed based on Elasticsearch, where data and events are mostly collected through several lightweight shippers. Data can also be collected through Logstash[7], an open server-side data processing pipeline that ingests data from a multitude of sources, which transforms and then sends them to Elasticsearch. The event captors activate accordingly when the appropriate event happens, query Elasticsearch, evaluate the results and push back the relevant information to the monitor for further evaluation.

3.2.2 FISHY and the Evidence Collection Engine

Currently the Evidence Collection Engine is targeting to facilitate the security assessments provided by the respective component of the SAP and with respect to the CIA standards. The Event Captor Module is initiated through the respectively REST calls from the Monitoring Module and communicates with the Elasticsearch via the respectively API which provides the following methods:

- *void getEvents(String eventsPath, Long groupID)*
  - **Description**: Declares the channel and the queue where the *event* captors are listening for input from the *monitor*
  - **Parameters**:
    - *eventsPath*: The events location
    - groupID: The assessment's unique ID
  - **Return value**: void

- *void generatedEvents(String eventsPath, Long groupID)*
  - **Description**: Pushes the events to RabbitMQ message broker from the *event captors*
  - **Parameters**:
    - *eventsPath*: The events location
    - groupID: The assessment's unique ID
  - **Return value**: void

[7] https://www.elastic.co/logstash/

For the sake of clarity, we summarize the calls in the following table:

Table 1 -Summary of the API calls of the Evidence Collection Engine module

Name of the method	Returning value	Description	Parameters
getEvents	*void*	Declares the channel and the queue where the *event* captors are listening for input from the *monitor*	▪ *eventsPath*: The events location ▪ *groupID: The assessment's unique ID*
generatedEvents	*Void*	Pushes the events to RabbitMQ message broker from the *event captors*	▪ *eventsPath: The events location* ▪ *groupID: The assessment's unique ID*

3.3 SACM

3.3.1 Monitoring the ICT infrastructure

The monitoring assessments are performed by SAPs Monitoring Module, a generic engine for checking violations of Event Calculus [1] formulae against a given set of runtime events and providing continuous evaluation of the security posture of FISHY's platform throughout its different layers.

The rules that need to be audited by the *monitor* are specified within security and dependability (S&D) patterns using an XML-based language called EC-Assertion. EC-Assertion is a first-order temporal logic language, based on Event Calculus (EC), primarily developed not only to represent but also to reason about actions and their effects over time. The basic elements of Event Calculus are *events* and *fluents*. An event in EC is specified as something that occurs at a specific instance of time and is of instantaneous duration. Furthermore, it may cause some change in the state of the reality that is being modelled while this state is represented by fluents.

To represent the occurrence of an event, EC uses the predicate *Happens(e, t, ℜ(t1,t2))*, which represents the occurrence of an event *e* that occurs at some time point t within the time range *ℜ(t1, t2)* and is of instantaneous duration. The EC predicate *Initiates(e, f, t)* signifies that a fluent f starts to hold after the event *e* occurs at time *t*. The EC predicate *Terminates(e, f, t)* signifies that a fluent *f* ceases to hold after the event *e* occurs at time *t*. An EC formula may also use the predicates *Initially(f)* and *HoldsAt(f, t)* to signify that a fluent *f* holds at the start of the operation of a system and that *f* holds at time *t* respectively.

EC-Assertion adopts the basic representation principles of EC and its axiomatic foundation and introduces special terms to represent the types of events and conditions that are needed for runtime monitoring. More specifically, given its focus on auditing the operation of software systems at runtime, events in EC-Assertion can be invocations of system operations, responses from such operations, or exchanges of messages between different system components. To represent these types of events, EC-Assertion defines a specific event structure that is syntactically represented by the event term *event(_id, _sender, _receiver, _status, _sig, _source)*. In this event term:

- **_id** is a unique identifier of the event.
- **_sender** is the identifier of the system component that sends the message/operation call/response.
- **_receiver** is the identifier of the system component that receives the message/operation call/response.
- **_status** is the processing status of an event (i.e., request(REQ) if the event represents an operation invocation and response(RES) if the event represents an operation response).
- **_sig** is the signature of the dispatched message or the operation invocation/response that is represented by the event, comprising the operation name and its arguments/result.
- **_source** is the identifier of the component where the event was captured.

The monitoring assessment results hold various monitoring-based parameters such as (a) the outcome of the monitoring process (Satisfaction if a monitoring rule was satisfied and violation elsewhere) and (b) the events involved.

3.3.2 Automation of the SACM

To provide security assurance and a certifiable view of each security assessment, the SAP aggregates the assessment results by executing an automated workflow comprised of eleven steps, as presented below:

1. Through the dashboard, the administrator selects to initiate one or more *security assessments* for auditing. In this step, the security assessments are described in a high level.
2. The Asset Loader Module pushes the additional information regarding the selected security assessments with a low-level description to the Monitoring Module. This low-level description may include the type and period (continuous or within time margins) of the evaluation and the type of asset that the security assessment refers to.
3. The Monitoring Module creates the respective audit instances for each selected security assessment.
4. The monitor sends information for each selected security assessment to the Event Captor Module to initiate the respective *event* captors.
5. Each audit instance starts listening for incoming events from the *event captors*.
6. The ELK stack is continuously augmented by the log shippers that are fetching events and data from the infrastructure.
7. The *event captors* query the ELK stack with a query generated specifically for the audited security assessment.
8. When a change is detected from the log shippers, the Event Captor pushes the information to the RabbitMQ Message Broker allowing communication between the *monitor* and the *event captors*.
9. The *monitor* decides whether the security metric is violated or fulfilled.
10. The SAP aggregates the assessment results to generate a **certifiable view** for each assessment.
11. The SAP dashboard is being updated with the assessment results that can be used as a certifiable view of the FISHY's platform's security posture.

3.3.3 SACM integration

As described before, the auditing component offers an Open REST API for establishing the auditing rules to be checked while it uses a relative API internally to communicate with the Evidence Collection Engine. Furthermore, the integration process is driven by the FISHY pilots needs and therefore we offer the capability to easily alter these REST APIs to support the pilots uses cases.

However, a series of generic REST APIs have been already formed to support an initial integration of SACM.

Three majors REST API calls have been implemented regarding the previous integration with the FISHY platform:

- Initiating auditing assessment. The POST method is described as follows,

  <u>/monitor/initiate/assessment-models/{assertionmodelID}/assessment-profiles/{assessmentprofileID}/projects/{projectID}/organizations/{organizationID}/asset/{assetID}</u>

- Stopping the auditing assessment instance with docker-instance scheduling involved. The POST method is described as follows,

  /monitor/stopservice/port/{monitorport}/

- Storing Certification Model as a blob file. The POST method is described as follows,

  /monitor/storeblob/

In addition, the Evidence collection Engine may parse events outside from the ELK approach that described in previous section. Currently, besides the Elasticsearch as primary log aggregator we are supporting the capability of reading event/log files through a message broker, based on RabbitMQ. Such approach will facilitate different needs of our FISHY pilots in case the latter are already using the ELK stack as their primary event collector.

4 SCM integration

4.1 Interactions

For the Security and Certification Manager (SCM) to be integrated into the FISHY platform, we consider the following interactions:

- Task 4.1 for the integration of EDC
- Task 4.2 for the integration of SACM
- WP3 due to interactions with Trust Manager
- WP5, given that this work package leads the implementation of the different components into the FISHY platform and communication with both IRO and SIA is needed.

4.2 SCM design and architecture

4.2.1 Introduction

As part of the SCM integration, the following components are required:

- A backend, to orchestrate SCM
- A frontend (or interface), for the interaction with the user
- Data exchange / communications, essential for the engagement with other components
- Analytics / data analysis, to analyze, improve and take advantage of communications amongst the different components that play a role in the FISHY architecture (as far as SCM is concerned).

4.2.2 Use cases

Prior to defining the architecture, we had to describe the use cases for the user, that is, the actions that can be performed by him/her. This is an important step to:

- Better understand and define how the actor, that is, the user should interact with the system.
- Serve as an introduction to the sequence diagrams presented later in this section.
- Accurately shape SCM architecture.

The following figure shows all WP4 use cases:

Figure 4 - WP4 end-user use cases

So as per the use case diagram, the actions that the user can perform include the following:

- Managing security controls
- Managing policies
- Translate high-level policies into low level configurations. This is done in two-steps:
    a) Refinement of HLP, which generates abstract policies and topology graph.
    b) Translation an abstract policy into a Network Security Function (NSF). The NSF translates an abstract policy into the low-level configuration.
- Check ICT system information including:
    a) Audit information and procedures
    b) Metrics, KPIs and other possible performance indicators
    c) System status: as a primary function, the purpose of FISHY is to oversee the status of the monitored infrastructure continuously.
- Translate requirements into policies
- Analyze the consequences of certain factors, that is, assess cascading effects.
- Check certification models
- Validate ICT audit procedures

4.2.3 Sequence diagrams

4.2.3.1 Manage controls

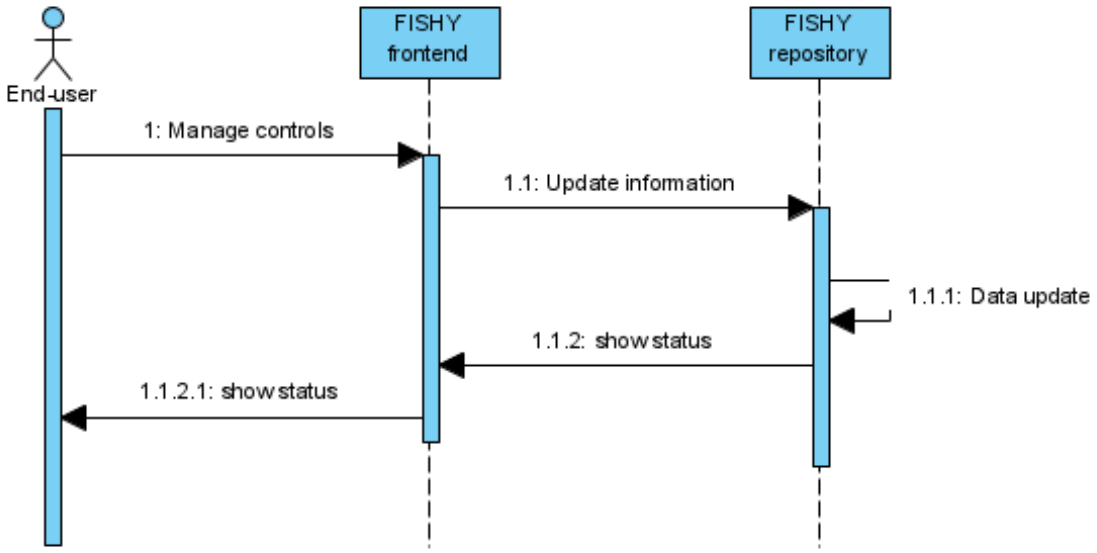

Figure 5 - Manage controls sequence diagram

4.2.3.2 Manage policy

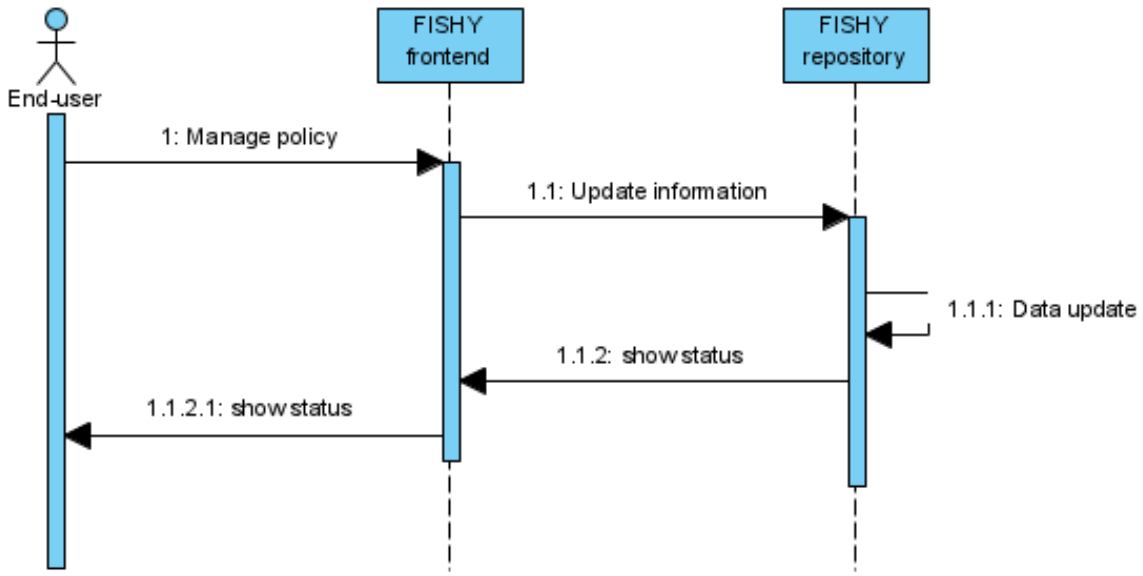

Figure 6 - Manage policy sequence diagram

4.2.3.3 Translate HLP into low level configuration

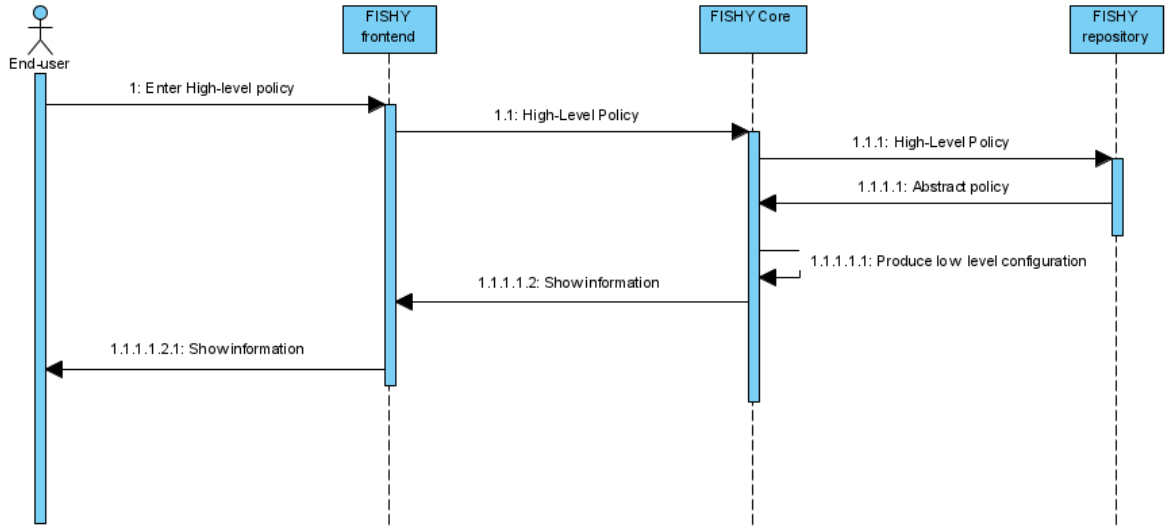

Figure 7 - Translate HLP into low level configuration sequence diagram

4.2.3.4 Check certification models

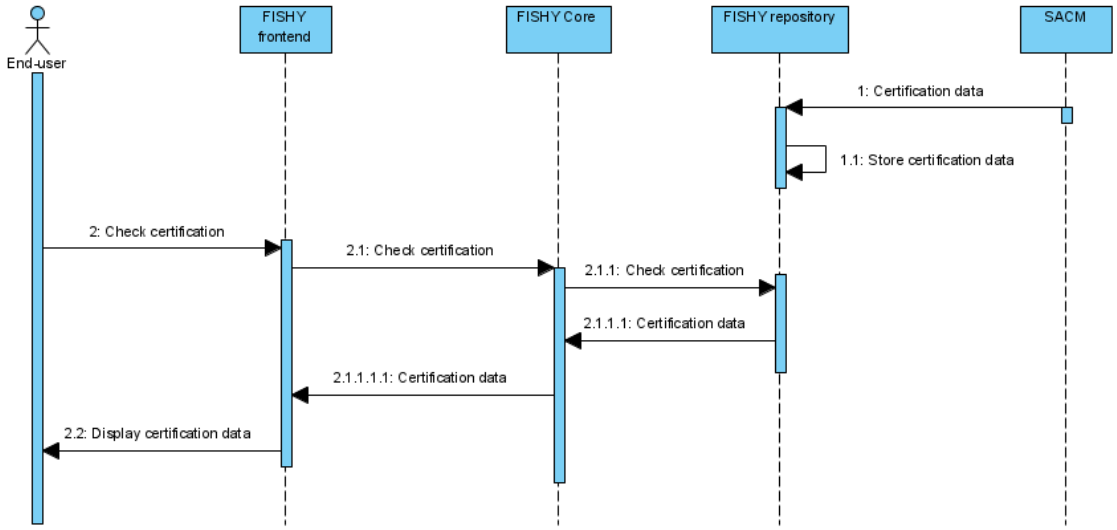

Figure 8 - Check certification models sequence diagram

4.2.3.5 Analyse cascading effects

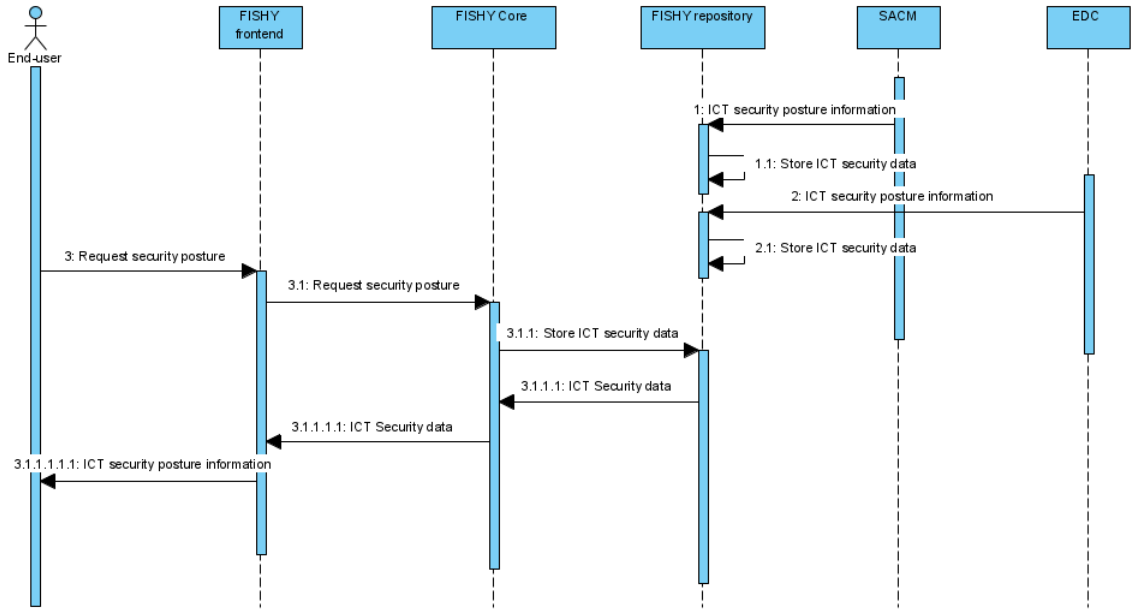

Figure 9 - Analyse cascading effects sequence diagram

4.2.3.6 Obtain ICT system audit information

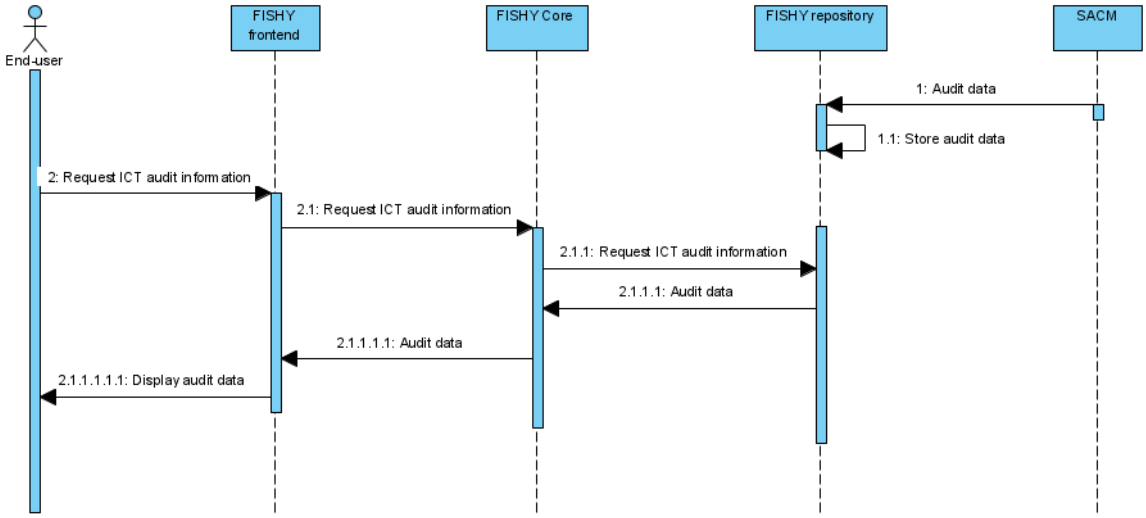

Figure 10 - Obtain ICT system audit information sequence diagram

4.2.3.7 Check ICT system metrics, KPIs

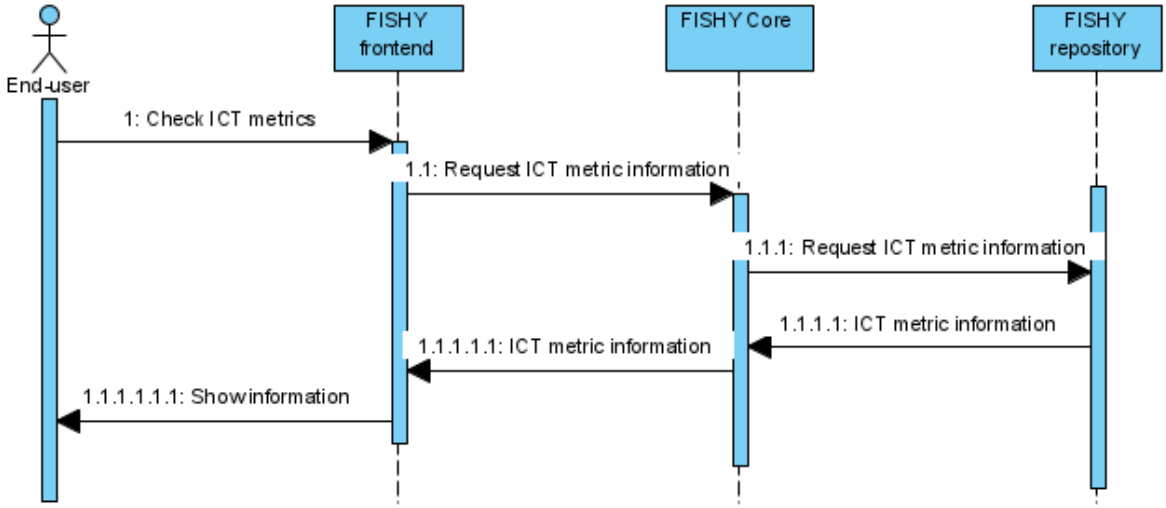

Figure 11 - Check ICT system metrics, KPIs sequence diagram

4.2.3.8 Check ICT system status

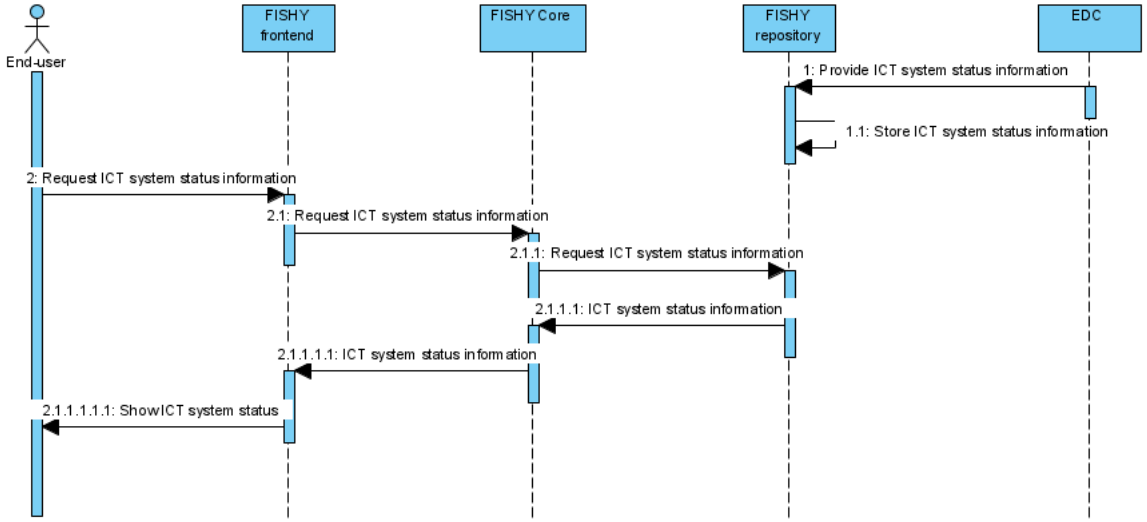

Figure 12 - Check ICT system status sequence diagram

4.2.3.9 Request ICT system security verification

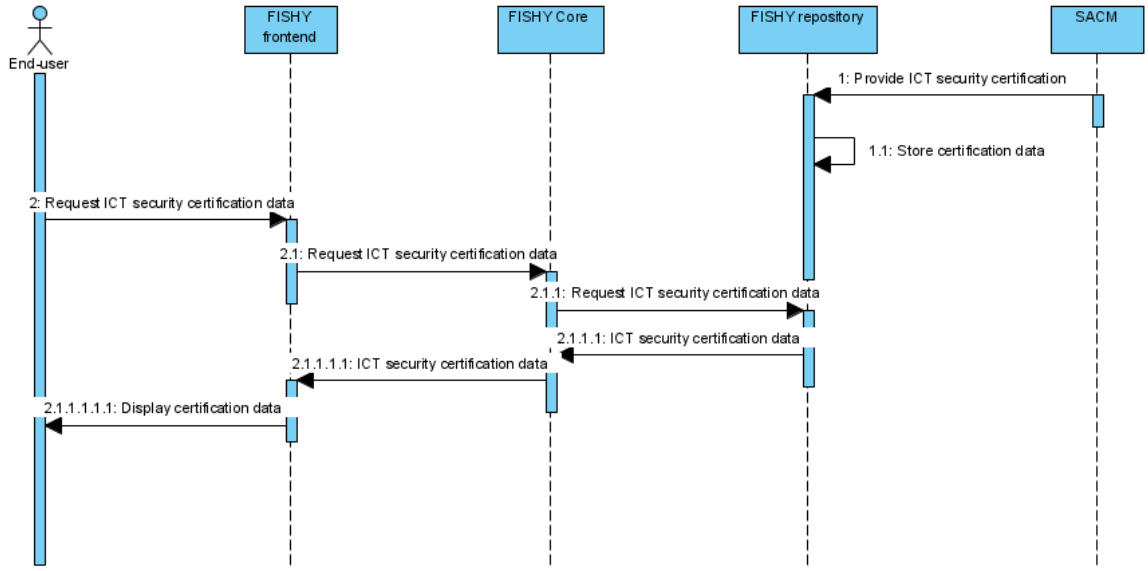

Figure 13 - Request ICT system security verification sequence diagram

4.2.3.10 Check & validate ICT system audit procedures

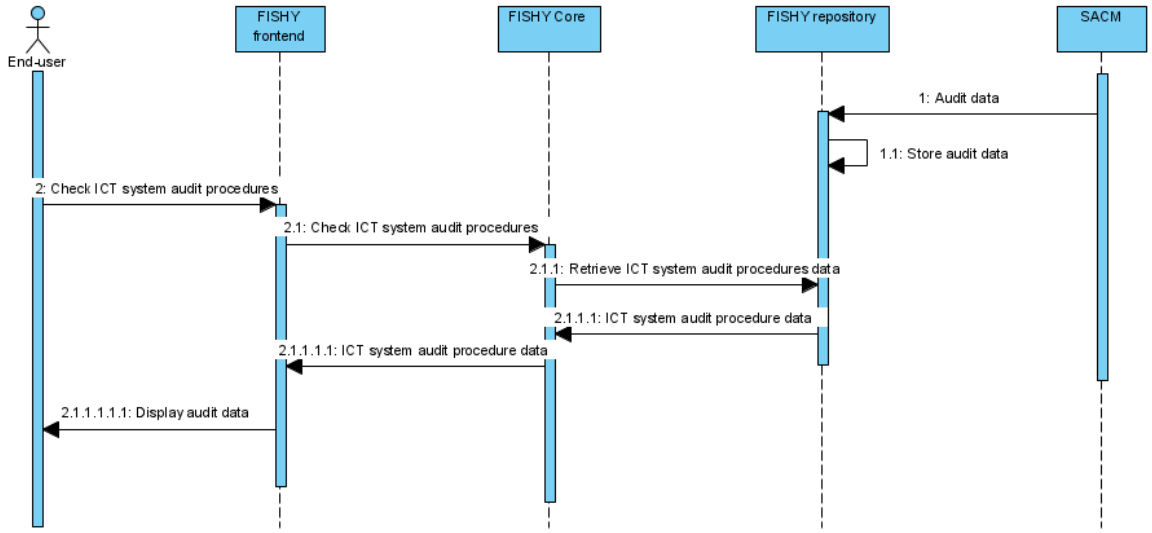

Figure 14 - Check & validate ICT system audit procedures sequence diagram

4.2.4 Architecture

The design is depicted in the following architecture diagram:

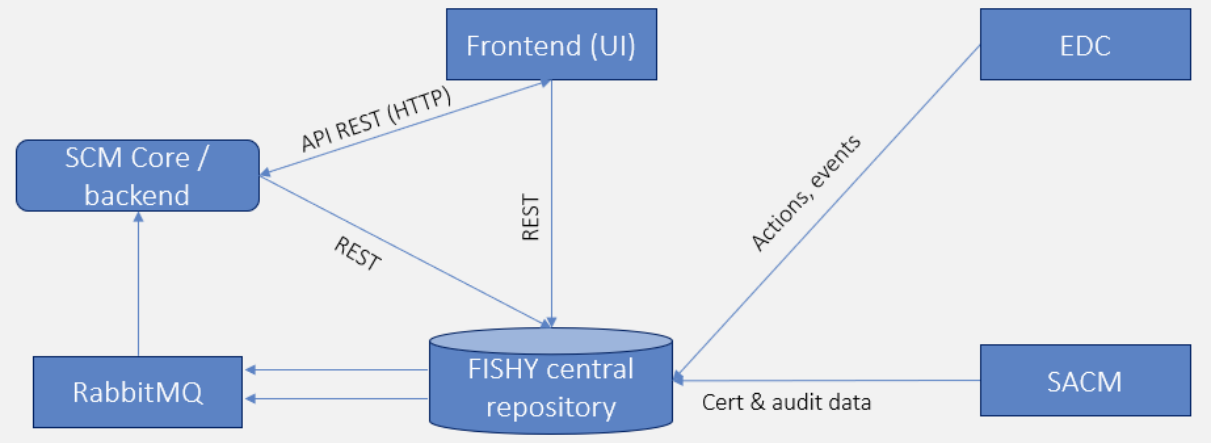

Figure 15 - Proposed architecture for SCM

4.2.5 SCM Core / Backend

The backend is the component that orchestrates the behaviour of SCM. Sometimes it is defined as a layer for data access. It supports all the logic needed to coordinate SCM functions. In addition, the backend acts as the internal SCM architecture while, at the same time, making all the components work together properly.

In the case of FISHY T4.3, the backend is expected to perform the following functions:

- Coordinates and orchestrates the SCM logic.
- Provides an answer to all frontend requests, that means feeding information to the frontend and, thus, the end-user. The way of communication will be a REST API.
- Feed other components and answer requests. The information provided by other FISHY components such as the EDC or the SACM will be:
    a) Stored in the central repository and
    b) Gathered in the corresponding RabbitMQ queue where the backend can pick it.

It is still to be defined what language will be employed for the development of the backend. One possibility is to use Python.

4.2.6 Frontend (UI)

The user interface (UI) or, simply, the interface, is a key aspect of any architecture. The UI is required for the interaction with users. As a more general idea, the interface encapsulates all inputs and outputs of the system. Besides, the interface defines a boundary, so it can transform the inputs provided by the user and route the data to the core or, in this case, to the SCM of the FISHY platform, defining the behaviour of the system.

The interface is part of the design of the SCM. We will use a two-step approach: determine first the components that are expected to take part in the architecture, and then, define the interfaces that will support and interact with these components. If the system is structured in various levels, interfaces at the lower ones are requested to provide more details.

The frontend and the UI are not the same concepts. However, for the sake of simplicity, we are assuming that the frontend to be developed in the FISHY T4.3 includes the UI so, it will provide interaction with the end-user. Functions to be accomplished by the frontend include:

- Help the user and try to make the user experience smoother.
- Provide a point where the user can feed information to FISHY.
- Provide information to the user.

The language to develop this frontend is still to be defined.

4.2.7 Data exchange and communications, the Knowledge Base

Data exchange may refer to a wide range of information flows, such as the following ones:

- A pub/sub mechanism where a producer delivers content to be consumed by the subscribers. A notification will indicate when new data is available.
- A system pushes data to another system or component.
- A component or system requests, that is, pulls information from another source.
- A central controller manages or, more accurately, orchestrates the implementation of information flows and processes.
- There is an event handler with the purpose of making decisions on how and when the information should be put to circulate in the system.
- Documents are also an example of data flow and information exchange, while also emails make information flow.

Concerning SCM implementation, communication, and data flow suggest the development of APIs for the exchange of data, that is:

- API for communication with IRO component (WP5 component)
- API for communication with SIA (WP5 component)
- Data exchange with the central FISHY repository or knowledge base (KB):
    - Both the EDC and the SACM are expected to provide information to the repository.
    - The data is to be stored in the repository and pushed to the respective RabbitMQ queues.
    - Finally, the data in the RabbitMQ will be gathered by the SCM backend.
- The backend will send information to the frontend by a REST API.

In FISHY T4.3, we will adopt one of the most convenient ways to implement APIs, i.e., by means of Swagger[8], an opensource toolset that helps the developer when documenting the APIs. This will be the way for API development.

4.2.8 Analytics

The very first idea of analytics is related to big data and data mining from raw information. Analytics is performed to acquire knowledge and take better decisions, since it allows getting insight of the systems and processes and, thus, enhancing performance. From a business perspective, the outcomes of applying analytics include improving customer experience and UX.

In addition, analytics applied to the system often result in a better operative and higher efficiency, which is a bonus. Analytics allows for a deep understanding of how the data flows across the architecture and how it should best be stored.

[8] https://swagger.io

Data analytics can be performed in various ways, including predictive or prescriptive analysis, and diagnostic and descriptive analysis.

Concerning SCM implementation, analytics should be specifically limited to data analysis. The gathering and analysis process considers:

- Both the information sent and received from the FISHY platform. This includes data originated by FISHY WP3, WP4 components, given that the FISHY architecture is about the interconnection of various components to define the whole system.
- Data managed internally, that is, amongst SCM components in the architecture described before.

Therefore, data flow analysis entails a certain transcendence since all components rely on a frequent exchange of information about the targeted infrastructure.

The process of analytics in the SCM implementation may require different options. We are considering using the R language[9]. This is an opensource programming language employed for statistical analysis and data mining. R is not tied to any platform and can be easily enhanced by means of lots of packages (10000+) developed and maintained by the community. R seems to be a feasible option for data mining, although there are other possibilities such as Apache Spark (open source), Power BI (commercial), or Tableau (commercial).

4.2.9 Additional remarks

Finally, we would like to make some final remarks regarding the architecture:

- Concerning storage, the possibility of including its own, dedicated storage for SCM was discussed with the Consortium. However, given that the FISHY platform owns central knowledge storage, there is no need to implement additional storage and would just mean to make it redundant. Therefore, SCM follows the agreements reached amongst project partners to make use of the KB of the FISHY platform.
- Regarding RabbitMQ, it has been proposed to use two queues. However, the architecture might also work with one RabbitMQ queue. The idea behind using two is to clearly differentiate the information provided by EDC and SACM.
- The number of APIs required for the data exchange is tentative. It means that new APIs could be needed. With the implementation of the SCM, the real needs for the component in terms of APIs will be clearly defined.

[9] R: The R Project for Statistical Computing (r-project.org)

5 Conclusions

The FISHY platform relies on the integration of various heterogeneous components that conform to the ecosystem. While the purpose of overseeing the monitored infrastructure is clear and shared amongst all components, the plan to achieve the integration into the platform must be carefully addressed and approached.

The diverse nature of the components and solutions aggregated to FISHY is part of its value. However, this also makes the implementation harder, than to other platforms where the architecture is made up of more homogeneous modules.

This deliverable 4.2 has described major components that take part in the FISHY WP4 implementation, that is, to aggregate SCM to FISHY, being also EDC and SACM key factors in the whole implementation. Besides, the need for a proper alignment with both WP3 (Trust Manager) and WP5 (integration package) entails a great challenge.

This document has also detailed the integration of the SCM components for the IT-1 iteration in the framework of the FISHY project and, thus, can be used as an input for deliverable D4.4 about IT-2 integration of the SCM component. In addition, also D4.3 (SCM components design and implementation, IT-2) could take advantage of the progress and work described in the present document.

Finally, regarding the following steps, we must consider:

- Defining the technologies to implement various components such as the language to develop the backend and frontend.
- Determining the number of APIs required as part of the SCM implementation.
- The integration of EDC and SACM into FISHY and the alignment with WP5.
- The testing and refinement of the integrated platform.
- At a long term, the release of an IT-2 prototype.

References

[1] Shanahan, M, *The Event Calculus explained*, https://www.doc.ic.ac.uk/~mpsha/ECExplained.pdf, retrieved 2021-09-24.

[2] [FISHY] - D3.2 Trust Manager IT-1 integration, Eva Marin-Tordera,2021.

[3] [FISHY] – D4.1 Security and Certification Manager components design and implementation (IT-1), Cataldo Basile, 2021.