Title:	*Trustworthiness mechanisms for Smart IoT Systems - Final version*
Authors:	*Erkuden Rios (TECNALIA), Eider Iturbe (TECNALIA), Angel Rego (TECNALIA), Saturnino Martinez (TECNALIA), Borja Urquizu (TECNALIA), Hui Song (SINTEF), Rustem Dautov (SINTEF), Phu Nguyen (SINTEF), Arda Goknil (SINTEF), Anne Gallon (EVIDIAN), Christophe Guionneau (EVIDIAN), Olivier Verdun (EVIDIAN), Samuel Mamuye (EVIDIAN), Jean-Christophe Durieu (EVIDIAN)*
Editor:	*Erkuden Rios (TECNALIA)*
Reviewers:	*Sergio Jiménez Gómez (INDRA), Jon Colado García (INDRA), Jean-Yves Tigli (CNRS), Nicolas Ferry (CNRS), Franck Chauvel (SINTEF)*
Identifier:	*Deliverable # D4.3*
Nature:	*Other*
Date:	*30th November 2020*
Status:	*Final*
Diss.	*level: Public*

Executive Summary

This deliverable provides the final version of the trustworthiness mechanisms for Smart IoT Systems of the ENACT toolkit. This includes the technical description of the final methods and prototypal developments made for the three main enablers in WP4: the Robustness and Resilience Enabler (DivEnact), the Security and Privacy Monitoring and Control Enabler and the Context-Aware Access Control Enabler. In addition, this document provides the final status of the WP1 requirement coverage related to these Enablers.

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Members of the ENACT consortium:

SINTEF AS	Norway
EVIDIAN SA	France
INDRA Sistemas SA	Spain
FundacionTecnalia Research & Innovation	Spain
TellU AS	Norway
Centre National de la Recherche Scientifique	France
Universitaet Duisburg-Essen	Germany
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia
Montimage	France
Beawre	Spain

Revision history

Date	Version	Author	Comments
26-08-2020	V0.1	Erkuden Rios	TOC proposed
04-11-2020	V0.2	Hui Song, Rustem Dautov	Added new section 4.3
04-11-2020	V0.2.1	Erkuden Rios, Angel Rego	Added new section 3.1.2 and updated 4.2.2
10-11-2020	V0.2.2	Hui Song, Rustem Dautov	Updates to section 2 and 1.4
10-11-2020	V0.2.3	Anne Gallon, Christophe Guionneau, Olivier Verdun, Samuel Mamuye, Jean-Christophe Durieu	Updates to section 4.2.1 and 1.4
13-11-2020	V0.2.4	Eider Iturbe, Saturnino	Updates to all section 4.1 and 1.4
13-11-2020	V0.3	Erkuden Rios, Eider Iturbe	Internal review version
20-11-2020	V0.4	Erkuden Rios, Eider Iturbe, Hui Song, Anne Gallon	Internal revised version partially addressing reviewers' feedback.
27-11-2020	V0.6	Erkuden Rios, Eider Iturbe	Final Internal revised version addressing all reviewers' feedback.
30-11-2020	V1.0	Erkuden Rios	Final version

Contents

1 Introduction

1.1 Context and objectives

The WP4 in ENACT aims at developing methods and tools supporting the security, privacy, and resilience of Smart IoT Systems (SIS) throughout the DevOps process cycle (see Figure 1). Smart IoT Systems in ENACT refer to next generation IoT systems which need to perform distributed processing and coordinated behaviour across IoT, edge and cloud infrastructures, manage the closed loop from sensing to actuation, and cope with vast heterogeneity, scalability and dynamicity of IoT devices and their environments.

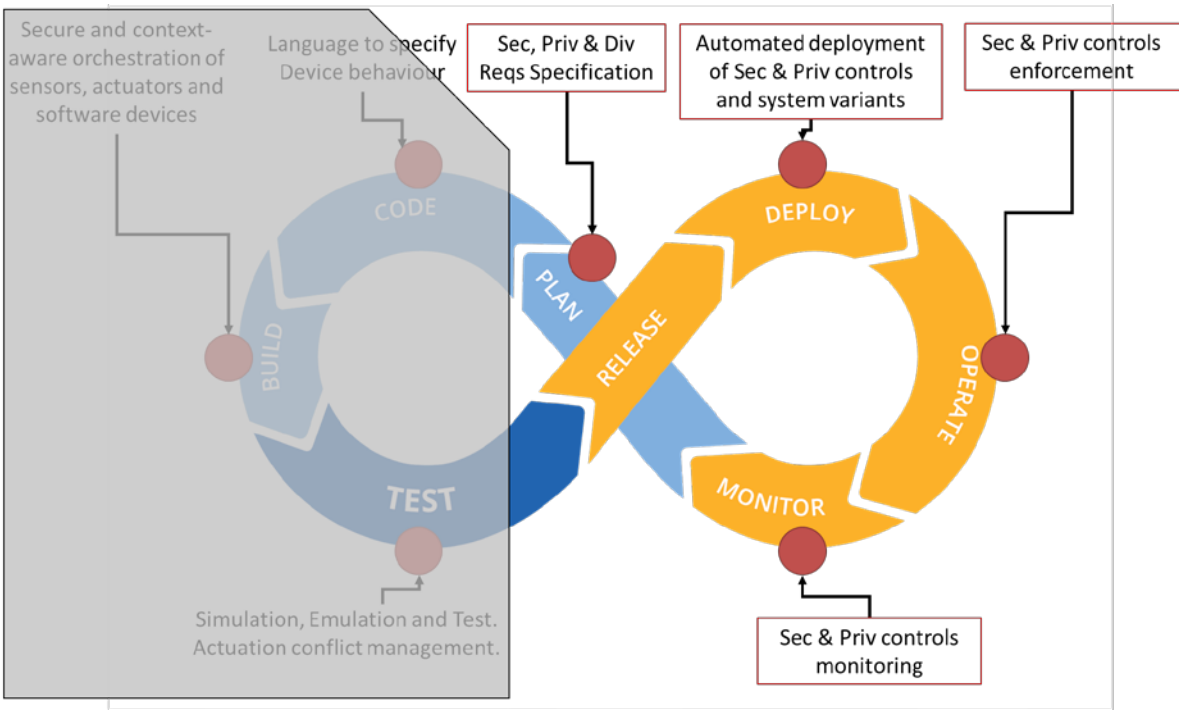

Figure 1. ENACT WP4 focus within DevOps cycle

As explained in D4.1, the WP4 deals with supporting **Security** (preservation of confidentiality, integrity, and availability), **Privacy,** and **Resilience of SIS**. In ENACT, SIS Resilience is based on enabling system components' software diversity to reduce the exposure of particular faults of the SIS to potential attackers as well as promptly recover from external perturbations.

The results of WP4 are shaped as three main enablers:

- **Robustness and Resilience Enabler:** In order to contribute to smart IoT systems trustworthiness, this enabler increases resilience of SIS by injecting or managing software diversity. This implies that each instance of a service has a different implementation and operates differently, still ensuring that its global behaviour is consistent and predictable. The enabler will automate the introduction and management of diversity in smart IoT systems, and in the following it will be referred to as ***DivEnact***.
- **Security and Privacy Monitoring and Control Enabler:** This enabler includes mechanisms and tools to control the security and privacy behaviour of IoT systems and to early detect anomalies by continuously monitoring security metrics defined during the project. This includes early reaction models and mechanisms that address adaptation and recovery of the IoT application operation in case of deviation from the expected behaviour of monitored security and privacy related metrics. Note that the security and privacy-aware behaviour of the SIS

complements other adaptation mechanisms to support SIS trustworthiness proposed by WP3 (see deliverable D3.2). Specific focus will be done on the confidentiality and integrity of data and services.

– **Context-Aware Access Control Enabler** for advanced access control and authorization mechanisms tailored to smart IoT systems. Today, no protocol can deliver dynamic authorization based on context for both IT (information technologies) and OT (operational technologies) domains. This enabler addresses this point by allowing to modulate the authorizations provided according to the assessed risk for a given context. Note that in D4.1 and D4.2 this enabler was explained as part of the Security and Privacy Monitoring and Control Enabler but we now present it as a separate enabler according to the packaging of ENACT tools described in WP5.

The deliverable D4.1 focused on the state-of-the-art and Use Case requirement analysis to derive the security, privacy and resilience mechanisms necessary to support security-by-design, privacy-by-design, resilience-by-design as well as monitoring and operational control of these aspects in SIS. Then, the deliverable D4.2 described the initial version of the methods and tools developed and how they covered the identified Use Case requirements.

The present D4.3 report is linked to the final delivery of the prototypes of the tools provided through the WP4 enablers. It focuses on presenting the achieved advances over the state of the art and the final coverage of Use Case requirements and presents the operation of the different tools and how they can be used in the ENACT framework to support SIS trustworthiness aspects.

1.2 Change log from deliverable D4.2

With the aim of delivering a self-contained description of the ENACT outcomes on security and privacy mechanisms for Smart IoT Systems, this final report builds upon previous description of the mechanisms.

This section provides the record of the updates and additions included in this document, the final ENACT Trustworthiness mechanisms for SIS, compared to the description of the intermediate version (deliverable D4.2 ENACT Trustworthiness mechanisms for Smart IoT Systems – First version).

Table 1: Overview of updates since D4.2

Change	Section in D4.2	Section in D4.3	Rationale
Updated collection of achievements according to the final work in WP4	Section 1.3	Section 1.3	Summarises the achievements offered by ENACT WP4 tools described in this document in regards with ENACT WP4 objectives on SIS system security and privacy support.
New section	N/A	Section 1.4	Summarises the support offered by ENACT Trustworthiness tools described in this document in regards with the properties of SIS system Trustworthiness.
Updated analysis with respect to final coverage of requirements.	Section 2	Section 2	The analysis describes now with respect to final coverage offered.

New section	N/A	Section 3.1.2	Explains the role of IoT threat analysis and its links with security and privacy controls of SIS.
Numbering of section	Section 3.1.2	Section 3.1.3	Numbering changed.
Updated description of security and privacy monitoring	Section 4.1	Section 4.1	Description of security and privacy monitoring reflects now the final architecture (section 4.1.1.2) and detailed design (section 4.1.1.3), which includes the implementation of the AI-based cyber incidents and anomaly detection service.
Updated restructuring of Operation support mechanisms description	Section 4	Section 4	The structure now presents each of the WP4 tools in different subsections. Section 4.1 corresponds to the Security and Privacy Monitoring and Control Enabler, Section 4.2 Context-Aware Access Control Enabler and Section 4.3 to Resilience support with DivEnact Enabler.
Updated description of the monitoring mechanisms, and included the detection and notification capabilities	Section 4.1	Section 4.1.1	The technical details of the final monitoring mechanisms and tool developed are described together with details of the artificial intelligence-based anomaly detection and holistic situational awareness through notifications and graph visualisation.
Updated description of the CAAC	Section 4.2.1	Section 4.2	The description now covers the whole capabilities of the CAAC Enabler. This includes: - In Section 4.2.2, description of a new infrastructure aimed to gather contextual information to deduce a risk level associated with a user. - In Section 4.2.3, description of contextual information management and risk computation. - In Section 4.2.4, consideration of risk levels as a dynamic attribute in the authorisation process. - In Section 4.2.5, description of the new CAAC test portal. - In Section 4.2.6, outline of main innovation of the CAAC developed.
Updated description of security and privacy controls in	Section 4.2.2	Section 4.1.2	The technical details of the final monitoring and control mechanisms developed and

IoT platform based on SOFIA SMOOL [1].			included in SOFIA SMOOL-based IoT platforms are provided.
Updated description of diversity support in DivEnact.	Section 4.3	Section 4.3.1 and Section 4.32.	Updates made to reflect the latest features and tutorial for DivEnact tool.
New section for new Fleet management support of DivEnact Enabler.	N/A	Section 4.3.3, Section 4.3.4, Section 4.3.5 and Section 4.3.6	New concept of "fleet deployment" based on the previous experiments of diverse-oriented deployment of IoT devices is described. Whole content in Sections 4.3.3 to 4.3.6 is new.
Updated conclusions and future work	Section 5	Section 5	Section with final conclusions and planned research lines beyond ENACT project end.

1.3 Overview and Achievements

The following table summarises the achievements of WP4 at the time of delivering D4.3, i.e. at the end of WP4 research and development work.

Table 2. Achievements of ENACT WP4 at the time of D4.3 delivery.

Objectives	Achievements so far and future work
State-of-the-art on IoT security, privacy and resilience	In D4.1 we conducted an extensive analysis of the state-of-the-art on approaches for security, privacy and resilience of SIS. We specifically focused on four topics: 1. IoT Security and Privacy challenges. 2. IoT Security- and Privacy-by-design. 3. IoT Security and Privacy assurance. 4. Software diversity as resilience mechanism in SIS.
Security and privacy-aware design and orchestration of IoT systems i) The languages and formalisms to enable the specification of the security and privacy requirements of smart IoT applications as part of the overall design, including the corresponding security and privacy metrics and probes allowing appropriate monitoring.	Description of possible mechanisms and procedures to express security and privacy requirements and controls in smart IoT systems design. Final design and implementation of Security & Privacy monitoring enabler is ready.

	Final formalisms to support to security and privacy specification in Orchestration and deployment models with GeneSIS is ready.
ii) Risk model characterizing potential security and privacy risks, considering both the characteristics of infrastructure devices and requirements of the smart IoT application (this will be integrated with WP2 risk driven orchestration and the decision support system for selection of the devices).	Addressed in D2.2
iii) Metrics of software diversity of individual services and the whole system.	Final diversity mechanisms already designed and first implementation ready as GeneSIS Orchestrator tool plugin.
Robustness, security and privacy enforcement in smart IoT systems	Final design and implementation of context-based authentication and authorisation of devices and services. Specification of the interfaces of the **Context-aware Access control tool** and description of the various ways this tool can be used to secure the IoT accesses, considering the context-awareness by taking into account the contextual information in a dynamic risk level computation. The context-awareness aspect of these mechanisms has been defined and implemented and final validation by Use Case providers will be reported in WP1. The **final design of diversity mechanisms** has been developed to be able to diversify IoT services, i.e., to automatically generate diverse versions of IoT services from the same ThingML model. In addition, research and prototyping has been conducted on the diversity controlling at runtime, with a diversifier tool named **DivEnact**. On top of that, we propose a new concept of **fleet deployment** for diversity-oriented automatic software deployment of large fleet of IoT/Edge devices and clarify its role in the overall DevOps lifecycle of applications across IoT/Edge/Cloud continuum. We implement a complete prototype tool, DivEnact, for fleet deployment in the DevOps of IoT applications. We conducted research on the novel way of automatic assignment of multiple deployment plans into large number of devices, as the key component in fleet deployment. The research approaches convert the software assignment

	tasks into constraint solving and resource assignment problems and employs advanced tools in the relevant domains to solve the problem. The implementations are integrated into the DivEnact tools. Final **design and implementation of the IoT Platform level security and privacy** mechanisms include security extensions of SMOOL (SOFIA platform) and enforcement mechanisms through application level agents to address the adaptation and recovery of the IoT application operation in case of monitored metrics deviation from the normal (risk under control) behaviour.
Security and privacy monitoring of smart IoT systems	**Final design and implementation** of the mechanisms and prototypes for **continuously monitoring the security and privacy behaviour** of IoT applications and early detect anomalies have been developed. The monitoring is able to capture data from different layers: network, system and application in order to enrich the detection and increase its accuracy. All these data are combined and correlated by the tool for advanced intrusion detection and anomaly detection. **Artificial intelligence** detection mechanisms combined with a multi-layer surveillance dashboard are offered by the tool, so as a holistic **situational awareness** of the cybersecurity status of the SIS and all its parts is enabled.

1.4 Support to trustworthiness in SIS

This section describes the characteristics of the support offered by ENACT WP4 tools to the five trustworthiness properties of SIS. The following table summarizes how each of the final tool prototypes developed in WP4 addresses each of the properties.

Table 3. ENACT WP4 methods and tools' support to trustworthiness in SIS at the time of D4.3 delivery.

WP4 tool	Security	Privacy	Reliability	Resilience	Safety
DivEnact	**Yes.** By diversifying software at the Edge, we implement a "moving target" defense strategy	**No.**	**Yes.** Diversifying software can lead to an increase in their reliability.	**Yes.** Supports failure recovery in a DevOps process	**No.**

S&P Monitoring and control	**Yes.** Monitors and controls confidentiality, integrity and availability of data	**Yes.** Monitors and controls confidentiality, integrity and availability of personal data	**Yes.** Monitors availability	**No.**	**No.**
CAAC	**Yes.** Confidentiality: ensure proper access control to the data	**Yes.** Ensure privacy in the data managed by Smart IoT Systems, since the control over the personal data is kept by their owner	**Yes.** Information is made available or disclosed only to authorized individuals, entities, or processes, in a controlled way.	**No.**	**No.**

In the following we outline the status of all the DevOps-oriented trustworthiness features of SIS that were planned to be delivered by WP4 at the end of the project (as described in the "Extra Document"[1]). At the time of the present document edition, all the promised features are delivered as summarized in the following tables, each table corresponding to one of the tools in WP4.

DivEnact:

Table 4. DivEnact features for trustworthiness in SIS.

Feature	Link	Status at M22	Status at M35
Generate device specific deployment models from templates	Section 4.3.2	Planned	Completed
Appoint a universal deployment for all devices	Section 4.3.2	Completed	Improved and revised with Added GUI and REST API
Appoint a subset of devices for preview	Section 4.3.2		
Promote preview deployment to production	Section 4.3.2		
Roll back devices	Sections 4.3.2, 4.3.3	Planned	Completed: support roll back for particular devices and all devices with a specific software version
Diversify the applications among identical devices based on the context/hardware conditions	Section 4.3.4, 4.3.5	Planned	Completed
Recover failed devices to the default or the last-working deployment configuration	Section 4.3.2, 4.3.3	Planned	Completed: support the default deployment on the fleet level and integrate the device-level recovery through GeneSIS

[1] "Extra Document": Plans for development and evaluation for 2020 and until the end of ENACT. Submitted to the Project Officer in January 2020 as an additional document to the planned deliverables.

Security and privacy Monitoring and Control tool:

Table 5. Security and privacy Monitoring and Control tool features for trustworthiness in SIS.

Feature	Link	Status at M22	Status at M35
Near real-time monitoring of the overall SIS	Sections 4.1.1.2 and 4.1.1.3	Completed	Completed and revised
Scalability for Security monitoring of SIS	Sections 4.1.1.3.1 and 4.1.1.3.2	Completed	Completed and revised
Security Monitoring in SMOOL SOFIA IoT platform	Section 4.1.2.2	Completed	Completed
Enhanced SMOOL SOFIA ontology	Section 4.1.2.1	Completed	Completed
SMOOL Security KP	Sections 4.1.2.2 and 4.1.2.3	Ongoing	Completed and revised
Advanced anomaly detection of SIS based on Artificial Intelligence	Section 4.1.1.3.6	Planned	Completed
Notification of events and alarms of detected issues	Section 4.1.1.3.5	Completed	Completed and revised
Situational awareness about SIS security (Dashboard)	Section 4.1.1.3.5	Completed	Completed and revised

Context Aware Access Control tool:

Table 6. CAAC features for trustworthiness in SIS.

Feature	Link	Status at M22	Status at M35
Device enrolment management	Section 4.2.4	Completed	Completed
Device authorization management	Sections 4.2.3, 4.2.4	Completed	Completed
Protected resource access management	Section 4.2.3, 4.2.4	Completed	Completed and extended with context-awareness capabilities.
Test portal and demo platform (first release)	Section 4.2.5	Ongoing	Completed and used in the eHealth demo Use Case in mid-term review.
Customizable post-authorization plugin	Section 4.2.2	Planned	Completed
Risk Server REST API	Sections 4.2.2, 4.2.3	Planned	Implemented and integrated into a global infrastructure for obtaining contextual data and provide CAAC with risk values. An interface allows external components to send contextual data into this infrastructure.
Context-aware risk computation configuration	Section 4.2.3, 4.2.5	Planned	Implemented through the demo platform.
Test portal and demo platform (second release)	Section 4.2.5	Planned	Completed and access provided to eHealth Use Case provider.

Large scale environments emulation tests	Section 4.2.5	Planned	Ongoing: simulation features are provided in the demo platform to test scalability.

1.5 Structure of the document

After the introductory section, the reminder of the document is structured as follows.

Section 2 analyses the coverage of the requirements of ENACT Use Cases with respect to SIS security, privacy and resilience, to demonstrate the degree of fulfilment by initial WP4 prototypes of those requirements.

Section 3 describes the different methods and mechanisms being offered as part of the ENACT solution to support SIS developers in creating trustworthy SIS, by providing security-by-design, privacy-by-design and resilience-by-design techniques to be adopted in SIS development.

Section 4 describes the different mechanisms being developed as part of the ENACT solution to support IoT system operation with respect to ensuring a secure, resilient and privacy-respectful behaviour.

Section 5 offers the conclusions of the work and describes the future work plans and further research lines beyond ENACT project end.

1.6 Acronyms and abbreviations

API	Application Programming Interface	JWT	JSON Web Token
BTE	Binary Tree Encryption	KP	Knowledge Processor
CAAC	Context-aware Access Control	OIDC	OpenID Connect
GDPR	EU General Data Protection Regulation	OS	Operating System
GUI	Graphical User Interface	OT	Operational Technologies
GW	Gateway	REST	Representational State Transfer
HTTP	HyperText Transfer Protocol	S&P	Security & Privacy
HTTPS	HyperText Transfer Protocol Secure	SIS	Smart IoT System
IDS	Intrusion Detection System	SSAP	Source Service Access Point
IDP	Identity Provider	SSH	Secure SHell
IoT	Internet of Things	TCP	Transmission Control Protocol
IP	Internet Protocol	TLS	Transport Layer Security
IPS	Intrusion Prevention System	XACML	eXtensible Access Control Markup Language
IT	Information Technologies	WAM	Web Access Manager
ITS	Intelligent Transportation System		

2 Analysis of Use Cases requirements coverage

The analysis of ENACT Use Cases carried out within WP1 produced a number of usage scenarios from where the requirements for the ENACT enablers were derived. Such analysis was complemented in WP4 with a dedicated questionnaire that inquired about different security and privacy aspects that clarified the needs of the Use Cases in terms of (personal) data protection both at rest and in transfer (see deliverable D4.1).

Together with the initial analysis of security and privacy aspects, the Use Cases identified a number of requirements related to the trustworthiness support in ENACT (collected in D1.1). From them, the requirements related to security, privacy and resilience will be used to evaluate the success of WP4 solutions. Table 7 summarizes the degree of fulfilment by initial WP4 prototypes of those requirements. In particular, the considered requirements refer to the following tools: (i) context aware access control, (ii) software diversity of IoT systems and (iii) privacy and security monitoring and control.

As it can be seen, in the eyes of the end-users in ENACT, all requirements are high or medium priority and all medium priority ones are recommended or nice to have features.

In the last column of the table, Coverage, we have indicated the degree of fulfilment of the requirement by the final version of the implementation of the corresponding WP4 Enabler, which has been updated since previous deliverable D4.2 and shows the final progress of the work.

It is worth noting that in some cases, the requirement is very specific to the particular Use Case IoT application and ENACT will not address it by means of an external enabler, but rather rely on the particular IoT application itself to resolve it.

However, for most requirements the software components or modules that have been developed in WP4 (Code Diversifier, Security and privacy Monitoring Enabler, Context Aware Access Control Enabler, etc.) have fully addressed the issue. See sections 3 and 4 to understand how each of the modules fits into the ENACT WP4 Enablers.

Table 7. Security, Privacy and Resilience Requirements Coverage by initial WP4 Enablers.

ID	Statement	Source[2]	Brief description	Priority[3]	Need[4]	How to Address in WP4/IoT app	Coverage
DO-4	***4. ENACT Trustworthiness Toolkit***						
DO-4.1	***4.1. Robustness & Resilience Enabler***						
DO-4.1.1	Gateway recovery and factory reset	2	There is a need to allow for resetting the Medical Gateway to factory default when something goes wrong and then get the GW operational after reset	H	M	Diversifier	**(Covered – It was only partly covered in D4.2)** DivEnact covers the bootstrap of gateway, which works both from scratch and as a recovery from factory reset. The actual reset currently requires device-specific mechanisms and end-user intervention. This will be the next step. The planned "next step" of device-specific mechanism is implemented by leveraging the blue-green deployment mechanism in GeneSIS, based on the integration between DivEnact and GeneSIS.
DO-4.1.2	Handle Medical Gateway failure situations	2	The Medical Gateway should quickly recover from being unresponsive. In addition to extensive and continuous testing this, includes features for handling the failure for example through remote access in a safe mode.	M	R	Diversifier	**(Covered – It was only partly covered in D4.2)** DivEnact deploys default applications into gateways automatically and remotely. The next step is to investigate and prepare a minimal "safe mode" deployment as a default deployment for the purpose of trouble shooting. This approach is now supported by DivEnact implemented in D4.3C
DO-4.1.3	Roll back configuration	2	In case a deployment of a new configuration fails. The GW should be able to rollback to the previous	H	M	Diversifier	**(Covered – It was not covered in D4.2)** Rollback is supported by DivEnact, in particular, it records the last deployment that was used successfully, and when rollback is required for a device, DivEnact pushes the recorded deployment to this specific device.

[2] Source - 1: ITS use case, 2: Digital Health use case; 3: Smart Building use case.
[3] Priority - H: High; M: Medium, L: Low.
[4] Need - M: Mandatory; R: Recommendation.

ID	Statement	Source[2]	Brief description	Priority[3]	Need[4]	How to Address in WP4/IoT app	Coverage
			configuration and notify the Operator				
DO-4.2.x	***4.2. Risk-Driven Decision Support Enabler***					Addressed by WP2.	
DO-4.3	***4.3. Security and Privacy Monitoring and Control Enabler***						
DO-4.3.1	Authentication	1	Authentication procedures are applied to treat every data packet.	H	M	IoT app[5]	Not to be addressed by WP4.
DO-4.3.2	Authentication invalid	1	A procedure to deal with invalid authentication of the elements and users must be designed.	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.3	Security not variable	1	The security measurements are not adapted if the system is running	H	M	S&P[6] Monitoring Enabler	**(Covered – It was already covered in D4.2)** The settings of the monitoring metrics can be set before the monitoring starts.
DO-4.3.4	Authentication levels	1	Several authentication levels would be designed.	M	R	IoT app	Not to be addressed by WP4.
DO-4.3.5	Attacks historical	1	An historical of that would be created.	M	R	S&P Monitoring Enabler	**(Covered – It was already covered in D4.2)** The prototype stores statistics and information on anomalies detected.
DO-4.3.6	Things and On Board GWs identification management	1	The Ids of the system elements must be checked.	H	M	IoT app	Not to be addressed by WP4.

[5] IoT app – IoT application (particular to the use case).
[6] S&P – Security & Privacy.

ID	Statement	Source[2]	Brief description	Priority[3]	Need[4]	How to Address in WP4/IoT app	Coverage
DO-4.3.7	Access security	1	The users must be authorized to access to the tool which manages the SW updates.	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.8	Orchestration Interface	1	The Monitoring enabler notifies the Orchestration of alerts related with a shift in some of the elements performance after processing the data gathered.	H	M	S&P Monitoring Enabler	**(Covered – It was only partially covered in D4.2)** The prototype is able to detect whether deployed security mechanisms are working properly.
DO-4.3.10	Alarm thresholds configuration	3	The Trustworthiness Monitoring enabler should enable the user to set the desired thresholds to raise cybersecurity alarms.	H	M	S&P Monitoring Enabler	**(Covered – It was only partially covered in D4.2)** The prototype allows to set desired thresholds on the metrics to raise alerts.
DO-4.3.11	Security enforcement	3	The Trustworthiness Monitoring enabler should work together with Trustworthiness Adaptation Enabler which helps reacting to attacks or incidents	H	M	S&P Control Enabler	**(Covered – It was only partially covered in D4.2)** Security Adaptation Enabler is now the Control Manager within the S&P Monitoring and Control Enabler. Some reactions are automated (e.g. at IoT Platform level) and some rely on humans (via the DevOps cycle).
DO-4.3.12	Protection of person sensitive data	2	Secure data management across IoT edge and cloud is severe as the system typically handle person sensitive data.	H	M	CAAC, S&P Control Enabler	**(Covered – It was already covered in D4.2)** The SMOOL IoT Platform has been extended to support encryption and other security controls. The Context-Aware Access Control tool provides access control for both users and devices.
DO-4.3.13	Monitoring and control	2	There is a need to do Real-time monitoring of a set of Medical Gateways and to receive proper notifications with useful information in case of errors.	H	M	S&P Monitoring Enabler	**(Covered – It was partially covered in D4.2)** The real time monitoring and notification functionality has been finalised in the S&P Monitoring prototype, even if the integration of the prototype in TellU Use Case has not been prioritized.

ID	Statement	Source[2]	Brief description	Priority[3]	Need[4]	How to Address in WP4/IoT app	Coverage
DO-4.3.14	Access control	2	Different users and roles should have different level of access. Need support for role based and user-based access control. It would also be interesting to look at context aware authorization (e.g., in an emergency the access may be different than in normal operation)	H	M	CAAC	**(Covered – It was only partially covered in D4.2)** The Context-Aware Access Control tool is an evolution of the authentication and authorization mechanisms provided by Evidian Web Access Manager (WAM) intended for the Internet of Things. It provides access control for both users and devices. The context-awareness aspect of these mechanisms is implemented through an extension the protocol OpenID Connect to also consider dynamic attributes such as contextual risk level.
DO-4.3.15	Authentication	2	Various kinds and levels of authentication need to be supported both at the edge side and cloud side. Support for two-factor authentication (or similar level) is mandatory for a set of scenarios in the digital health domain	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.16	Secure data transmission	2	Confidentiality, integrity, and authentication across IoT, edge and cloud is needed.	H	M	Secure protocols, S&P Monitoring Enabler	**(Covered – It was only partially covered in D4.2)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Even if it was not prioritized to integrate the S&P Monitoring tool in TellU Use Case, the analysis of monitored data to detect anomalies and notification functionality has been finalised in the S&P Monitoring prototype.

ID	Statement	Source[2]	Brief description	Priority[3]	Need[4]	How to Address in WP4/IoT app	Coverage
DO-4.3.17	Communication need to be trustworthy in the sense of reliability, availability, integrity and privacy	2	The trustworthiness aspects of communication within digital health is significant for example, in order to not miss any notifications or alarms, you should be always connected to support emergency situations when they occur, the integrity of data is severe, and privacy needs to be ensured as there is typically person sensitive data involved	H	M	Secure protocols, S&P Monitoring Enabler	**(Covered – It was only partially covered in D4.2)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Even if it was not prioritized to integrate the S&P Monitoring tool in TellU Use Case, the analysis of monitored data to detect anomalies and notification functionality has been finalised in the S&P Monitoring prototype.
DO-4.3.18	Monitoring, Diagnose information and failure detection	2	The system should continuously monitor system performance, suspicious behavior and failures. The monitored data should be analyzed to provide informative and understandable diagnosis	M	R	S&P Monitoring Enabler	**(Covered – It was only partially covered in D4.2)** The analysis of monitored data to detect availability anomalies has been finalised in the S&P Monitoring prototype.
DO-4.3.19	Full end-to-end security	2	Support for security across the IoT, edge and cloud space from the medical device, through the gateway and all the way to the target stakeholders (e.g., hospitals, Electronic patient journal etc.) is needed	M	R	Secure protocols, S&P Monitoring Enabler, S&P Control Enabler, CAAC.	**(Covered – It was partially covered in D4.2)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Even if it was not prioritized to integrate the S&P Monitoring tool in TellU Use Case, the analysis of monitored data to detect anomalies has been finalised in the S&P Monitoring prototype.

3 Design support to IoT system Security, Privacy and Resilience

This section describes the methods and mechanisms offered by ENACT as security-by-design, privacy-by-design and resilience-by-design techniques to be adopted in SIS development.

The ENACT support to security and privacy at design time is focused on: i) mechanisms for the specification of both the requirements of the SIS components with respect to these two aspects, and ii) mechanisms for the specification of the necessary controls (external or internal to the SIS) that ensure the requirements are met. The approaches proposed for privacy requirements and controls specification are the same as those of security requirements and controls specification, which eases the engineering of the SIS since both aspects are addressed similarly in the DevOps process. Therefore, Section 3.1 provides the description of the mechanisms proposed for security-by-design and privacy-by-design together. Finally, Section 3.2 describes the diversity mechanisms that will be developed for diversity-by-design techniques.

3.1 IoT Security-by-design and Privacy-by-design mechanisms

In this section we describe the different mechanisms offered by ENACT solution to address trustworthiness in SIS design from the point of view of security and privacy-aware design. First, we explain the process supported by ENACT for the SIS security and privacy requirements expression (section 3.1.1). Second, the link with ENACT support to threat and risk analysis developed in WP2 is described (section 3.1.2). Finally, we explain the procedure and mechanisms for identification and definition of the security and privacy controls in the SIS design (section3.1.3), with a special focus (section 3.1.3.1) on the support provided in GeneSIS modelling language to this aim.

3.1.1 IoT Security and Privacy requirements specification

In ENACT the SIS specification activity consists in the system modelling supported by GeneSIS language and tool (see D2.3 for more details). Therefore, ENACT proposes to address the system security and privacy requirements definition together with architectural and deployment requirements definition as part of the GeneSIS model of the system. To this aim, ENACT has gone beyond the recent advances on CloudML extensions [2] to describe security requirements at component level in distributed multi-cloud applications evolving them towards IoT systems. ENACT WP2 has enriched GeneSIS to include concepts for the specification of security and privacy requirements similar to those of CloudML oriented formalisms (for example Provided and Required Ports with Security Capabilities, see details in D2.3).

The GeneSIS language, similarly to CloudML, enables to capture system requirements at high-level of abstraction and independently of the actual providers and services used. GeneSIS as CloudML is Cloud provider independent and is service independent. The transformation from CloudML to GeneSIS models is fully possible, while the transformation from GeneSIS to CloudML models is possible but only for the Cloud parts of the system architecture, because some aspects of GeneSIS models (e.g. Controller, Devices, Hardware Capabilities, see details in D2.3) cannot be represented in CloudML.

Therefore, the description of security (availability, integrity and confidentiality) and privacy (personal data protection and security measures applied to personal data) aspects of IoT systems will be addressed in ENACT by specifying within the system GeneSIS models which are the security and privacy controls associated to the components, be they internal or external to the system. By internal components we refer to those developed or owned by the DevOps team developing the IoT system and whose full control is in hands of the team, whereas external components would be those services used by the IoT system (invoked, used as deployment platform, etc.) but whose control resides in a third-party service provider.

As described in deliverable D2.3, the identification of security and privacy controls required and provided by the components deployed in the IoT environment will require a previous risk analysis combined with the self-assessment of the internal components.

The **self-assessment** is a necessary step towards understanding which security and privacy measures are being implemented in the components under development. Different approaches of performing this analysis can be adopted, such as the one proposed by MUSA [3], where the component developers or product owners would go through a guided checklist indicating which security and privacy controls from standard control families are offered by the component or service under analysis. This assessment will therefore result in the list of controls already available in the component and thus not needed to be requested to any third-party. In MUSA, the NIST SP 800-53 rev 4 [4] security control family was used, but other control families can be adopted, according to the interest of the organisation, such as ISO/IEC 27002 [5], ISO/IEC 27017 [6], ISO/IEC 27018 [7], CSA CCM [8], etc.

Self-assessment is a best practice promoted in the community of security and privacy experts in different fields. At cloud layer of the IoT System, the STAR program by Cloud Security Alliance promotes the use of the Consensus Assessments Initiative Questionnaire (CAIQ) [9] that provides a set of Yes/No questionnaire for assessing the status of the controls in the Cloud Controls Matrix. In web application context, the Application Security Verification Standard (ASVS 4.0) [10] proposed by OWASP could be used as the basis of the security features check. Similarly, Berkley DB best Practices [11] may help in assessing database security features and correct implementation of countermeasures in very specific contexts.

The self-assessment is a per component process and once all internal components in the IoT system are assessed, the DevOps team will have a clearer picture of security and privacy posture of the components integrating the system in form of a well-structured collection of offered security controls and privacy controls, which will be an input to the Risk Assessment process in order to identify whether there is any missing control for the system security and privacy behaviour assurance.

It is interesting to note that from all existing standard security control families, only NIST control family in its latest version rev 5 Draft [12] and the ISO/IEC 27552 [14] (under development) include actual privacy controls related to the protection of personal data. Furthermore, in addition to distinguishing between privacy and security controls, NIST SP 800-53 rev 5 Draft [12] identifies also *joint* controls as those able to address system security and privacy requirements at a time. At this point, one should bear in mind that privacy, even according to the GDPR, is intrinsically related to security mechanisms applied in personal data protection by services or systems. Article 5.1(f) of GDPR, requires that personal identifiable information (PII) *shall be processed in a manner that ensures appropriate **security** of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures ('integrity and confidentiality')*.

For external third-party components or services, it is necessary to learn in advance which are the offered security and privacy controls or policies, which usually depend on the type of service and/or supporting device (e.g. the available encryption modes in a gateway, the authentication mechanism used by a sensor, if any, etc.). This information is generally provided by the service/component (or device) provider, even if it not usually described in form of standard security or privacy controls. It is the task of the DevOps team to try to homogenise the way the offered security and privacy mechanisms are specified for external components with that of the internal components, so the overall security and privacy posture of the SIS is understood and can be controlled easily.

3.1.2 IoT Security and Privacy threat and risk analysis

Once the IoT assets and already available controls are identified, the possible threats against the assets and the overall system need to be analysed in order to identify the required protections as either prevention mechanisms or reaction mechanisms. This means that it is necessary to evaluate the relevant threats and either include in the IoT system design a number of protections or controls against them (e.g.

define in the IoT system design the necessary mechanisms to deploy together with the components) or activate the mechanisms at IoT system operation, such as intrusion detection and data protection.

The IoT system threat analysis includes the study of which are the relevant threats for the SIS under study, which represent the potential attack scenarios against the system. Generally, all the threat analysis approaches involve the identification of assets, system boundary mapping and decomposition, threat identification and vulnerability identification [15]. Threat sources and/or events are usually identified first and then threat scenarios and threat processes are studied in detail. The Threat modelling report by the Department of Homeland security [16] proposes that the threat analysis of a particular system or environment begins by establishing its scope, architecture and technology components, as well as the types of adversaries and range of attack techniques to be considered (for instance, attacks on the supply chain might be considered within scope or excluded). Security perimeters, interfaces and flows are examined to characterize the attack surface.

Analysing potential defence mechanisms to counter attacks is far from being a trivial and usually is tackled by security and privacy experts in the DevOps team. In IoT scenarios, knowledge on Cloud security issues and measures adoptable is required, since IoT can be supported by Cloud resources at application layer [17][18]. Some control catalogues such as the MUSA security metric catalogue mentioned above can aid in this task. The threats in this catalogue are mapped to both controls that could be used to counter the threats and control metrics that facilitate the assessment of control performance. The controls in the catalogue adhere to NIST SP 800-53 taxonomy rev 4 [4] which aids the auditability of defences included in the system, since these controls are mapped to other standards such as CSA's CCM [8] and ISO/IEC 27002 [5].

As explained in previous section, the self-assessment of internal system components will aid in identifying the existing defences or controls so as to better decide upon defences missing in the system. Besides internal components, threat analysis shall also study the envisaged attacks to outsourced components together with the defences that would be requested to the external providers. This issue is highly relevant in IoT environments where some or all of the distributed sensors, edges, fogs or Cloud services, may be outsourced components or services. The selection of system defences will need to consider the controls available from external services which the system will use (e.g. IoT Edge services, Infrastructure-as-a-service – usually private -, Software-as-a-Service, etc.) and the affordable security expenses in third-party components.

Formal threat analysis is often supported by threat modelling engineering discipline that supports the identification, rationalisation and reasoning about threats, attacks, vulnerabilities and countermeasures that could affect an application or system. A threat model abstracts the system under analysis and supports the reasoning over different kinds of weaknesses, potential attacks, or hazards of the system. Threat models usually represent (parts of) the software components or devices in a system, the data flows between them and the trust boundaries in the system, so threat vectors identification is easier. Threat models in machine-readable format that can be processed by computer programs enable systematic threat analysis, as the one proposed in ENACT.

The threat analysis in ENACT framework is an intrinsic step of the **security and privacy risk analysis**, which includes the estimation of the likelihood of actually happening the transformation of threats into attacks or incidents, as well as the evaluation of potential damage or impact that they can cause on the IoT system. A description of a systematic approach to threat and risk analysis validated in the Smart Building SIS Use Case can be found in [19].

The **Risk analysis and evaluation** is supported in ENACT by the Risk Management tool (see full description in deliverable D2.2) which, starting from threat analysis over data flow diagrams, will allow the DevOps team to reason over system assets requiring protection, learn which are their vulnerabilities and identify potential cyber incidents and attacks against these assets. At IoT system level, the assets would refer to system components. Therefore, the risk assessment will identify the most severe cyber risks over the components according to the IoT system architecture, communications between the components and types of data exchanged. The result of this process will be the identification of the relevant cyber threats that need to be addressed and the possible mitigation means. As part of these mitigations, the required organisational procedures (e.g. the inclusion of privacy aspects in the data

quality management procedures, the implementation of Privacy Impact Assessments, the security and privacy trainings, etc.) and required system mechanisms may be identified (such as the adoption of access control, the inclusion of a vulnerability scanning, the monitoring of a specific interfaces, the de-identification of personal data, etc.).

After the threat analysis and the risk assessment is done, and the necessary treatments decided, the necessary security controls to use in the SIS will need to be specified by the DevOps team as part of the security-by-design and privacy-by-design activities.

In the following we describe the challenges of the specification of selected security and privacy controls at SIS design time, and detail the support offered by ENACT to integrate them in the SIS design through the help of GeneSIS.

3.1.3 IoT Security and Privacy controls specification

As explained above, different control families can be used to express the security and privacy controls of the systems in a standard way. These include but are not limited to NIST SP 800-53 rev 5 Draft [12], ISO/IEC 27002 [5], ISO/IEC 27017 [6], ISO/IEC 27018 [7], CSA's CCM [8], etc. These catalogues aid the formal specification of required and/or offered (provided) capabilities of the IoT System as a whole and/or its individual components and the organisation providing it/them. In those cases where personally identifiable data is processed by the IoT system or any of its components, privacy controls need to be adopted.

As the major exponent of fine-grained controls catalogue publicly available for free and internationally adopted, the NIST SP 800-53 rev 5 Draft [12] collects 1026 controls identified as *security* controls, privacy controls and *joint* controls (which serve either security or privacy or both). A total of 161 privacy related controls are identified by NIST SP 800-53 rev 5 Draft [12] from 12 different areas or groups (named *control families* by NIST), namely: AC - Access Control, AT - Awareness and Training, AU - Audit, CA - Continuous Assessment, CM - Configuration Management, CP - Contingency Planning, IA - Identification and Authentication, IP - Individual Participation, IR - Incident Response, MP - Media Sanitization, PA - Privacy Authorization, PL - Planning, PM - Project Management, RA - Risk Assessment, SA - System and Services Acquisition, SC - System and Communications, SI - System and Information Integrity. From these, 60 controls are *privacy* controls and 101 are *joint* controls.

NIST also distinguishes between organisational controls ("O" – a control implemented by a human in the organization through nontechnical means) or system controls ("S" – a control typically implemented by an organizational system through technical means) or controls implemented by either or the combination of both nontechnical and technical means ("O/S"). The privacy related "S" and "O/S" controls identified by NIST rev 5 Draft are only 12 and 14 respectively, summing up a total of 26 technically implementable privacy controls. Therefore, according to NIST, most of the means for tackling with privacy assurance in the systems, reside at the organizational or procedural level, not at system level. In the following table, we summarize the system level controls related to privacy, trying to stick as much as possible to the NIST guidance on the controls. In the table, column 3 identifies privacy controls marked as "P" and joint controls marked as "J", while system controls are marked as "S" and controls that can be addressed with technical and/or non-technical means are marked as "O/S".

Table 8. System level controls related to privacy

Control ID	Control in the NIST SP 800-53 Revision 5 Draft	P vs. J[7]	S vs. O/S[8]	Summary
AC-16(1)	Security and Privacy Attributes | DYNAMIC ATTRIBUTE ASSOCIATION	J	S	Both security and privacy related properties or characteristics (attributes) of
AC-16(2)	Security and Privacy Attributes | ATTRIBUTE VALUE CHANGES BY AUTHORIZED INDIVIDUALS	J	S	

[7] P=Privacy control and J=Joint control
[8] S=System control and O/S= Organization and/or System control

AC-16(3)	Security and Privacy Attributes | MAINTENANCE OF ATTRIBUTE ASSOCIATIONS BY SYSTEM	J	S	data subjects and data objects should be considered.
AC-16(4)	Security and Privacy Attributes | ASSOCIATION OF ATTRIBUTES BY AUTHORIZED INDIVIDUALS	J	S	
AC-16(5)	Security and Privacy Attributes | ATTRIBUTE DISPLAYS FOR OUTPUT DEVICES	J	S	
AC-16(8)	Security and Privacy Attributes | ASSOCIATION TECHNIQUES AND TECHNOLOGIES	J	S	
AC-16(11)	Security and Privacy Attributes | AUDIT CHANGES	J	S	
AU-12(4)	Audit Generation | QUERY PARAMETER AUDITS OF PERSONALLY IDENTIFIABLE INFORMATION	P	S	Audit of query parameters within systems for datasets that contain personally identifiable information.
PA-3(2)	Purpose Specification | AUTOMATION	P	S	Automated mechanisms may be used to support records management of authorizing policies and procedures for personally identifiable information.
SC-7(24)	Boundary Protection | PERSONALLY IDENTIFIABLE INFORMATION	P	O/S	Ensure that personally identifiable information is used or transmitted only in accordance with established privacy requirements.
SC-16	**Transmission of Security and Privacy Attributes**	J	S	Associate organization-defined security and privacy attributes with information exchanged between systems and between system components.
SI-4(25)	System Monitoring | PERSONALLY IDENTIFIABLE INFORMATION MONITORING	P	O/S	Automate the monitoring of (a) unauthorized access or usage of personally identifiable information; and (b) the collection, creation, accuracy, relevance, timeliness, impact and completeness of personally identifiable information.
SI-6	**Security and Privacy Function Verification**	J	S	Verify the correct operation of not only security but also privacy functions of the system and notify and react in case anomalies are detected.
SI-15(1)	Information Output Filtering | LIMIT PERSONALLY IDENTIFIABLE INFORMATION DISSEMINATION	P	O/S	Limit the dissemination of personally identifiable information to organization-defined elements identified in the privacy risk assessment and consistent with authorized purposes.
SI-18	**Information Disposal**	P	O/S	Disposal or destruction of information (originals, copies, archived records, including system logs) that may contain personally identifiable information.
SI-19	**Data Quality Operations**	P	O/S	Perform operations on personal identifiable information to ensure its quality in terms of accuracy, relevance, timeliness, impact, completeness and de-identification.
SI-19(1)	Data Quality Operations | UPDATING AND CORRECTING PERSONALLY IDENTIFIABLE INFORMATION	P	O/S	
SI-19(2)	Data Quality Operations | DATA TAGS	P	O/S	
SI-19(3)	Data Quality Operations | PERSONALLY IDENTIFIABLE INFORMATION COLLECTION	P	O/S	
SI-20	**De-Identification**	P	O/S	

SI-20(1)	De-Identification | COLLECTION	P	O/S	Personally identifiable information shall be removed from datasets when it is not (or no longer) necessary to satisfy the requirements envisioned for the data.
SI-20(2)	De-Identification | ARCHIVING	P	O/S	
SI-20(3)	De-Identification | RELEASE	P	O/S	
SI-20(4)	De-Identification | REMOVAL, MASKING, ENCRYPTION, HASHING, OR REPLACEMENT OF DIRECT IDENTIFIERS	P	S	
SI-20(5)	De-Identification | STATISTICAL DISCLOSURE CONTROL	P	O/S	
SI-20(6)	De-Identification | DIFFERENTIAL PRIVACY	P	O/S	
SI-20(8)	De-Identification | MOTIVATED INTRUDER	P	O/S	

The control IDs that include a number between brackets refer to control enhancements which complement basic controls. Therefore, most of the privacy controls that can be addressed at system level are enhancements of other basic controls. Following this analysis, a total of 11 controls are addressed by technical measures (at system level) and of those, only 5 controls are basic controls and not enhancements: SC-16 Transmission of Security and Privacy Attributes, SI-6 Security and Privacy Function Verification, SI-18 Information Disposal, SI-19 Data Quality Operations, SI-20 De-Identification.

System level security and privacy controls described above are implemented in form of security and privacy mechanisms such as two-factor authentication, encryption, anonymisation, etc. which work together with system components and which can in turn be deployed in and managed from different SIS layers. As explained in the following section, ENACT has advanced in modelling languages and tools which support the expression and enactment of SIS models which include security and privacy mechanisms to be deployed and configured together with SIS components.

3.1.3.1 IoT Security and Privacy controls specification in GeneSIS

In this subsection, we present the language and tool support in GeneSIS for specifying IoT security and privacy controls in the deployment models of smart IoT systems (SIS) and deploying them together with the components of SIS. We show below some representative examples of the main features that GeneSIS supports for deploying security and privacy mechanisms. The features described in this subsection are derived from the collaboration between WP2 and WP4. More details of the GeneSIS tool and different features for supporting the deployment of IoT applications together with security and privacy mechanisms can be found in D2.3.

A main requirement for GeneSIS deployment model regarding support for security and privacy specification is that any software component can require to be deployed together with a specific security/privacy component that provides a certain security and privacy capability. GeneSIS can choose the security and privacy controls from the catalogues of security and privacy controls mentioned in Section 3.1.3 above. We basically organise the security and privacy controls into two main catalogues as shown in Figure 2. The first catalogue contains the ENACT-driven security and privacy controls that are tailored for the advanced supports regarding trustworthiness. The other catalogue consists of other security and privacy controls in the literature that can also be used by the GeneSIS. Figure 2 shows how the two catalogues used as input of GeneSIS. Driven by the Risk Management process, a DevOps team working on the deployment model using GeneSIS can select the security and privacy controls with required capabilities from the catalogues for risk treatment. GeneSIS allows the chosen security and privacy controls to be specified for being deployed together with the other components of SIS.

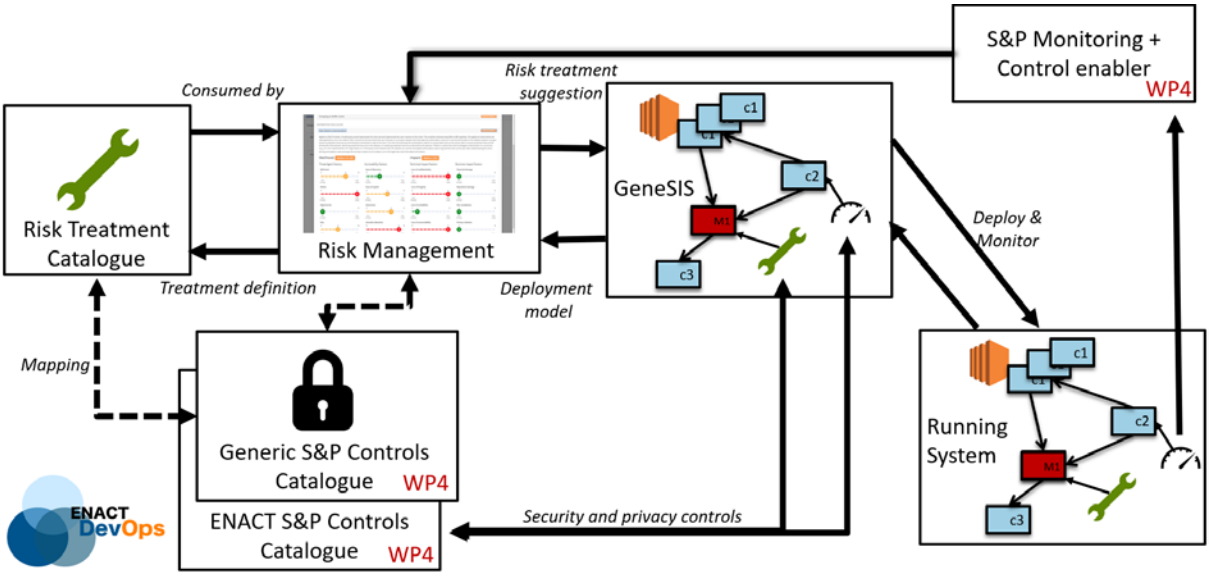

Figure 2. The security and privacy controls of WP4 as input for GeneSIS

In D2.3, we have described the current support of GeneSIS for deploying security and privacy controls together with the IoT software components of a SIS. For example, an IoT Energy Efficiency application that reads sensor data must only do so according to an access control policy, or context-based access control mechanisms. It is important that the IoT applications can only receive the sensors data they are allowed to access and can only send commands to the actuators they are allowed to control, in a dynamic context. For example, for security (and privacy) reasons, the IoT Energy Efficiency application can receive sensors data about temperature and light, but not live video from cameras. The IoT Energy Efficiency application can control heating system or window blinds, but not in case of fire alarm. For privacy reasons, camera data cannot be sent out of the gateway that directly connects to the camera. To address such security and privacy requirements for the IoT Energy Efficiency application, a DevOps team working on the IoT Energy Efficiency application can select and adapt the security and privacy controls with capacities satisfying the requirements, configure them with required policies and use GeneSIS to specify them to be deployed together with the components of the IoT Energy Efficiency application.

The examples of context-based access control and privacy policy enforcements discussed above are implemented at application level, normally engineered for a specific smart IoT system. The enforcement of security and privacy policies will be in place after GeneSIS deploys the required security and privacy mechanisms together with IoT applications. Therefore, the main requirement for GeneSIS is the following. When deploying the IoT applications, we also need to deploy security and privacy mechanisms together with policies that must be enforced for those IoT applications. From the deployment point of view, GeneSIS supports specifying 1) the capabilities (security/privacy) of each component, and 2) the ports associated with components. The port specification is useful for security mechanisms such as a Security Gateway that only exposes certain ports for controlled accesses to protected sensors or actuators. Figure 3 shows the concepts of ports and capabilities of components that are supported by GeneSIS. With these supports, each IoT component that requires a certain security or privacy control can only be 'linked' to components providing the necessary capabilities using a *Communication* link, via the provided ports. When GeneSIS deploys a security or privacy component/control, it means configuring it, including with security or privacy policies if any.

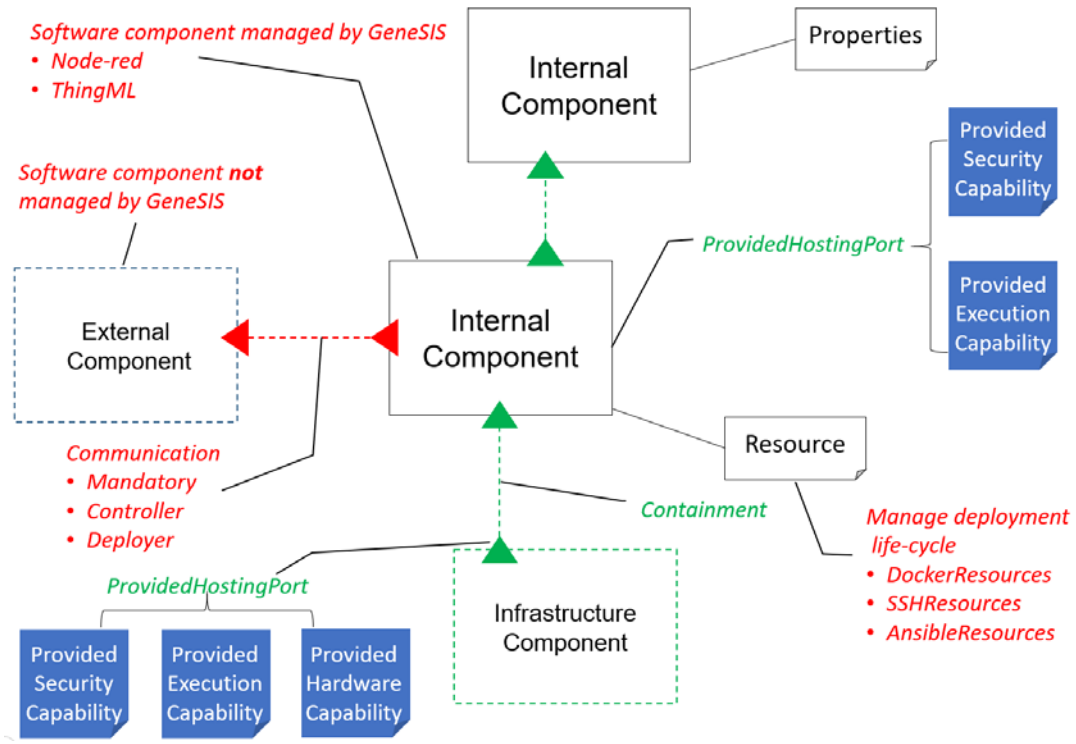

Figure 3. The generic concepts of components in GeneSIS with capability specification support

The main requirement presented above is the core feature for GeneSIS to support the deployment of security and privacy mechanisms. More "advanced" requirements are also considered by GeneSIS. For example, location constraints such as how far is the host of a security module from the sensors or actuators might be expressed. Such constraint reflects the fact that a security module may need to be a local node to a gateway to guarantee the performance of the authentication process. Moreover, a software component may have a constraint to be deployed only two “steps” away from the data source or actuator. For instance, this may be due to a privacy policy specifying that this specific software component that processes sensitive data cannot be deployed more than two network levels away from the data source.

Main innovation. The idea of specifying security aspects in the deployment model is implemented in the latest work of GeneSIS, and the details can be found in the Deliverable 2.3 – Section 4.4. The work contributes to the research community on automatic deployment by including first-class modeling elements in the high-level, normally functional, description of system architecture. The main achievement is published in the recent GeneSIS paper:

- Nicolas Ferry, Phu H Nguyen, Hui Song, Erkuden Rios, Eider Iturbe, Satur Martinez, Angel Rego, 2020, Journal of Object Technology, Volume 19, Issue 1.

It is worth noting that the focus of the recent work is in the direction of specifying security and privacy mechanisms as components in the deployment model, instead of as properties of the ports. We will keep this direction as the future work of GeneSIS.

3.2 IoT Diversity-by-design mechanisms[9]

(*This section has been reported in D4.2 and remained here for the sake of completeness)*

The ENACT IoT Diversity-by-design tool consumes as inputs and outputs some of the design-time artefacts in the ENACT project, i.e., the deployment model specified in the GeneSIS modelling language and the behaviour model specified in ThingML. In particular, the tools take as input a single deployment

[9] Source code of this tool can be found here: https://github.com/SINTEF-9012/thingml-diversifier

or behaviour specification and generate multiple diverse specifications. Within a DevOps context, it is important and necessary to keep the diversity generation fully automatic, instead of relying on developer's manual effort to main diverse systems (such as the traditional N-Version Programming approach [20]). Developers can focus on a single line of code to achieve frequent iteration and the diversification tool, as part of automatic building step, will generate diversified version automatically.

Automated diversity is a promising means of mitigating the consequences of a security breach. However, current automated diversity techniques operate on individual processes, leveraging mechanisms available at the lower levels of the software stack (in operating systems and compilers), yielding a limited amount of diversity. In this section, we present a novel approach for the automated synthesis of diversified protocols between processes. This approach builds on a) abstraction, where the original protocol is modelled by a set of communicating state machines, b) automated synthesis, applying mutation operators onto those protocols, which produces semantically-equivalent, yet phenotypically-different protocols and c) automated implementation of these protocols through code generation.

Mass-produced software applications denote clonal applications, with thousands or millions of identical siblings. Think of, for example, a popular mobile application installed on millions of mobile phones, or software embedded into a widely used connected device. To mitigate the risks of such large mono-cultures, diversity is typically automatically introduced either in a generic way, typically at the OS level, oblivious of the actual logic and semantics of the software, or in some very specific places, typically low-level libraries reused across applications, in order to improve security. This leaves most of the actual business logic unchanged, unaffected by the diversity. In addition, diversity often affects individual processes, but leaves the communication between processes intact.

A more holistic approach to diversity is challenging. Consider a typical client-server application, where multiple clients interact with a server and where each client has a different implementation and a different way of communicating with the server. This would significantly hinder a hacker, be it a human being or a machine, when attempting to generalize an attack through all possible protocols. This would make large-scale exploits a time-consuming and costly endeavour for hackers. Yet, the engineering, e.g., the production, maintenance and integration, of such levels of diversity raises several challenges. How to ensure that each implementation still behaves as specified? How to ensure that each client is still able to communicate with the server, without information loss or distortion? How to ensure that different clients are fundamentally (i.e., sufficiently) different and not merely cosmetically different? How to keep the development and operation costs of a diversified system significantly lower than the cost of mitigating large scale attacks?

We have seen that abstraction, synthesis and automated implementation can yield a convincing solution to introduce a wide diversity into protocols, for example between a device and a gateway, or a web/mobile app and a server [21]. This approach:

- abstracts protocols into a) a structural view describing the messages to be exchanged, and b) a behavioural view based on state machines describing how those messages are exchanged between the participants, including sequencing and timing.
- combines and applies a number of atomic mutations to this protocol model, yielding a large number of diversified protocols, which operate differently, still with the same semantics.
- automatically implement protocols, diversified or not, by generating fully operational code targeting C, Go, Java and JavaScript, able to run on a wide range of platforms.

Our empirical assessment indicated that this approach implies a reasonable overhead in terms of execution time, memory consumption and bandwidth, fully compatible with the requirements of “mass-produced” software. We also showed that this approach could generate a significant amount of diversity. Our assumption was that this diversity would contribute to the diversity-stability hypothesis, i.e., this would make the whole ecosystem more robust by making it less likely for an exploit to propagate to the whole population. In other words, if the protocol between a specific client and the server could be observed, analysed and eventually understood, this would not systematically imply that all other diversified protocols could be understood following the very same procedure.

In this section, we briefly describe the mechanisms and the corresponding tools we developed to automatically generate the diverse protocols. Technical details and the experiment results can be found in our conference paper [21].

Our approach relies on ThingML for the specification of protocols. ThingML provides a way to formalize the messages involved in protocols, in a comparable way to what Protocol Buffer proposes. In addition, ThingML provides a means to formalize the behaviour of protocols through state-machines. ThingML specifications are both human-readable and machine-readable, which makes it possible to analyse protocols at a high-level of abstraction and to fully automate the implementation of those protocols through code generation. In the next sub-section, we present relevant aspects of ThingML on our motivating example.

We model communication protocols as a set of communicating state-machines, encapsulated into components. A protocol typically involves two roles: 1) a client, i.e., a device, a web-browser, or a mobile app, and 2) a server i.e., a gateway or a cloud back-end. The clients and the server need to agree on a common API. Since communication is typically asynchronous in a distributed system, the common API is specified as a set of messages. Next, this API is imported by the client component and the server component, and the messages are organized into ports.

The ultimate goal of our approach is to diversify the wire image of protocols [22]. Diversifying the wire image of protocols basically means shuffling the sequence of bytes exchanged over the network e.g., turning the payloads while ensuring the interoperability between the client and the server. This comes with a number of constraints:

- The first byte of each block must be an identifier for the message encoded by this block. Though this is not an absolute requirement, it is a commonly accepted practice as it allows to implement, or generate, efficient parsers with no need for extra memory and back-tracking.
- The order of messages should be preserved.

The model-based diversification process itself is depicted in Figure 4. The diversifier is a 1-to-n, endogenous transformation. It takes a ThingML protocol model as input and generates *n* ThingML protocol models as output. A key benefit of this design choice is that the whole ThingML tool-chain can be reused as-is on those diversified protocols. The ultimate output of this process is a set of binaries, for any of the languages supported by ThingML, implementing the diversified protocols. Optionally, these binaries can be automatically packaged as Docker components for facilitating their deployment.

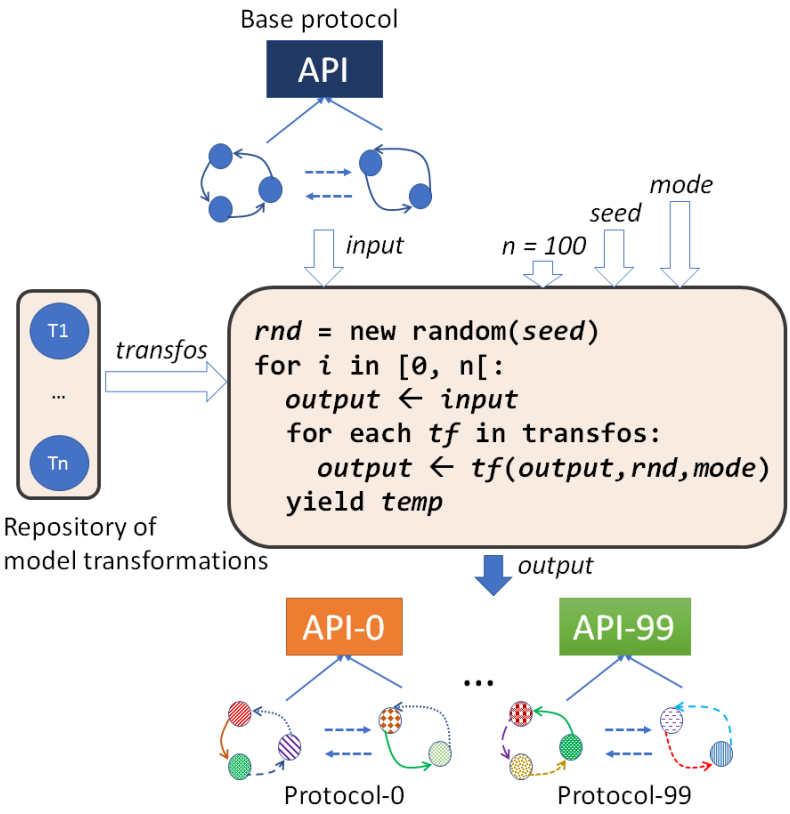

Figure 4. Model-based diversification process

The diversification process is configured according to the following parameters:

1. Seed: the seed for the random number generator used internally by the diversifier. Given one seed and one input protocol, the diversifier produces repeatable outputs.
2. Transformations: the sequence of model transformations to apply on the input model. Each transformation takes a valid ThingML model and produces more ThingML models, all of which are valid but different from the original one. The technical details of these transformations can be found in a published paper [15].
3. Mode: Most transformations introduce a random choice. Such a choice can be:
    - Static: in this mode, the diversifier generates different implementations for the input protocol. However, all the diversity is fixed by the diversifier itself - i.e., the protocol will not evolve over time.
    - Dynamic: in this mode, the diversifier still generates different implementations. In addition, some diversity still remains open at runtime, so as to allow the protocol to change over time (within some boundaries) and act as a moving target defence.
4. The number of diversified protocols to be generated.

The automatic generation of both client and server-side code is an essential feature of our MDE process for the diversification of communication. Here, we rely on the compilers available as part of the ThingML framework. The diversified protocol models being “plain old ThingML specifications”, the existing compilers. This way we can generate code in C (for resource-constrained microcontrollers or Linux), Java, JavaScript (for browsers or Node.JS) or Go for both the client and the server sides.

The design-time software diversification provides inputs to the runtime diversity control realized by the tools GeneSIS (see Deliverable 2.3) and DivEnact (see Section 4.4 in this document D4.3). In short, the

synthesis approach in this section generates various versions of software which will be deployed into the fleet of IoT devices, in order to achieve the diversity of entire fleet.

Main innovation. This approach provides an innovative solution towards the automatic synthesis of software diversity in the communication of components within IoT systems. The idea, solution and evaluation results are published in the paper below.

- Morin, B., Høgenes, J., Song, H., Harrand, N., & Baudry, B. (2018, October). Engineering Software Diversity: a Model-Based Approach to Systematically Diversify Communications. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (pp. 155-165).

We will keep our research in this direction after the ENACT project, aiming at generic and executable diversification synthesis tools for IoT software.

4 Operation support to IoT system Security, Privacy and Resilience

This section describes the support offered by WP4 Enabler prototypes to SIS operation with respect to ensuring a secure, resilient and privacy-respectful behaviour.

While monitoring support will be offered by the so-called Security and Privacy Monitoring and Control Enabler, the Context-Aware Access Control (CAAC) Enabler is dedicated to controlling the access to IoT system resources based on context attributes. SIS resilience support is provided by the Robustness and resilience Enabler, the so called DivEnact, which has been designed with a special focus on the large scale IoT systems, i.e., systems that comprise large number of sub systems, each of which may serve a different customer. The following subsections correspond to the technical description of each of the Enablers in WP4.

First, in section 4.1, we explain in detail the mechanisms offered in ENACT for ensuring that the security and privacy properties designed in the Development phase of the DevOps process are actually addressed in the SIS and work properly, i.e. the control mechanisms offered in Operation for the assurance of security and privacy capabilities of the SIS. From all the possible security and privacy aspects in a SIS, the focus of the work in ENACT is the assurance of confidentiality and integrity of the data, be it personal data (privacy) or non-personal data (security). Within this section, we first describe the provided monitoring solution, and then the control supported in the IoT Platform layer for IoT systems that use the SOFIA middleware.

Second, section 4.2 presents the new mechanism developed for context-aware access control to enable in the IoT systems fine-grained resource access policies associated to context conditions and risk levels in the SIS.

Finally, in section 4.3 we provide details of the new features of DivEnact Enabler to support the diversity-oriented fleet deployment and operation of SIS.

4.1 Security and Privacy Monitoring and Control Enabler

4.1.1 Security and Privacy Monitoring mechanisms

Addressing security and privacy assurance in IoT systems is a challenging task due to the complexity of its multi-layer nature (thing, edge, Cloud, network, etc.) and the heterogeneity of system components from system to system. In WP4 the research has focused on how to effectively offer to DevOps teams the means to be able to ensure the continuum of security and privacy properties of the system, while keeping the mechanisms independent from system particular implementation as much as possible.

To this aim, WP4 has designed and developed the Security and Privacy Monitoring and Control Enabler as support to the tasks of controlling (personal) data protection and secure communications all along the operational life of the SIS. The first step in the control is the continuous monitoring of the status of the system security and privacy. The Monitoring part of the Enabler is described in this section 4.1 while the Control part is described in the next section.

As it will be seen, for the Monitoring service in ENACT we have leveraged a number of state-of-the-art technologies which we have integrated together to offer a complete solution whose main innovation resides in three major axes:

- the **richness of the anomaly detection** by correlation of **multi-layer data** from the SIS. Through exploiting the data captured by multiple probes or monitoring agents deployed in network, application and system layers of the SIS, the developed anomaly detection offers a holistic view of situational awareness in the SIS and enable the detection of sophisticated attacks or incidents, allowing for easier identification of root-cause.

- the **flexibility and extensibility of the architecture** to address the needs of different types and sizes of SIS. In cases where it is not possible to deploy all types of monitoring agents, because of unreachability of a layer (network, system or application) the solution design enables to use only the types adoptable and still detect anomalies with the information available. The design allows also to focus on specific types of anomaly events when desired.
- the **scalability** of the solution. By relying on Big Data technologies, the solution guarantees that the Monitoring tool is fully elastic and able to rapidly scale in large-scale IoT system scenarios where hundreds or thousands of monitoring agents need to be deployed.

4.1.1.1 Purpose

According to multiple reference organisms such as NIST or ENISA, an imperative activity applied to an IoT system should be the Security Monitoring activity [23]. It includes protocols and data formats that enable the continuous, automated collection, monitoring, verification and maintenance of software, system and network security configurations, and provide greater awareness of vulnerabilities and threats and the overall status of the security of the system.

The Security and Privacy Monitoring tool in ENACT allows the IoT system operator to monitor the security and privacy status of the IoT system at different layers. The tool will capture and analyse data from multiple and heterogeneous sources such as raw data from network, system and application layers, as well as events from security monitoring components such as Intrusion Detection System/Intrusion Prevention System (IDS/IPS). All the data will be processed and displayed in a common dashboard, which includes processed information in form of alerts, statistics and graphs. The holistic dashboard makes it easy for DevOp teams to understand security status of the SIS and promptly react accordingly.

4.1.1.2 Architecture

Figure 5 shows the architecture details of the Security & Privacy Monitoring and Control Enabler. Specifically, the monitoring related components are represented in blue and purple, while the control related components are represented in green in the figure. The latter ones are described in more details in section 4.1.2.

The ENACT Security & Privacy Monitoring Enabler, in short, Monitoring Enabler, captures raw data from the SIS through multiple distributed probes, named *agents*. The distributed monitoring mechanism relies on three types of monitoring agents that can retrieve raw data from the SIS: (i) **network agent**, which captures network traffic data as well as identifies security events; (ii) **system agent**, which can capture data related to activities of processes on the devices of the SIS; and (iii) **app agent**, which can capture operational data of the SIS and also generates security events. The app agent in ENACT has been implemented as a special mechanism to work with SMOOL IoT Platform that will be used in the ENACT Smart building Use Case. There are two types of complementary app agents based on SMOOL IoT Platform addressing different scopes of security of the SIS. The first one of the app agents, called *Security KP*, is continuously listening to all communications in and out through the SMOOL IoT Platform. The second app agent, called *security aware KP*, is a customized application agent that is hosted in the IoT device and works with specifics of the application level. See section 4.1.2 for more information on SMOOL based app agents.

The Monitoring Enabler inputs are further complemented by an open source **Network Intrusion Detection System (NIDS),** which also captures network data and generates security events based on pre-defined rules. These pre-defined rules are settled by connecting to open source and/or commercial databases with knowledge of the well-known cyber security attacks. Moreover, the rules can be enhanced by the Monitoring Enabler user at any time; the only restriction is to follow the common format of defining the rules [38]. The NIDS is integrated within the network agent and it works at the edge level identifying security events.

In addition to the mentioned inputs, the Monitoring Enabler can also acquire events from other application layer security and privacy agents deployed in the SIS. For instance, the external security policy checking service described in section 4.1.2.2could send its events to the Monitoring Enabler.

All the IoT data captured and the security events generated by the Monitoring Enabler agents or the Security & Privacy Control Enabler agents are sent to a **streaming bus** that has a two-fold objective. First, the bus allows to acquire all the data in streaming and avoid a bottleneck at next phases of pre-processing of the IoT data. And second, it allows to send back the pre-processed IoT data and handle publish-subscribe threads related to the data handled by the streaming bus (pre-processed IoT data and events generated by different agents and components of the Security and Privacy Monitoring and Control Enabler).

After the acquisition of the IoT data and generated events, the next phase is to parse and do the required processing of the acquired data. The **Data Capturing and Parser** component is responsible of importing the IoT data and events published in the Streaming Bus and storing it into to the **IoT data storage** infrastructure. The Data Capturing and Parser component can also execute further processing over the IoT data and send it back to the Streaming bus. The latter feedback to the Streaming bus aims to provide the **Anomaly detection** component at prediction phase and to the **rule-based IDS** with near real-time data.

Furthermore, the Anomaly detection component can gather pre-processed IoT data from the IoT data storage at training phase. The Anomaly detection component will be trained using different Machine Learning algorithms and different models will be created for the prediction phase. The objective is to complement the existing IDS with other anomaly detection mechanisms for cyber security threats.

Finally, the **Monitoring dashboard** displays all acquired data and generated events in a user-friendly manner in form of alerts, statistics and graphs.

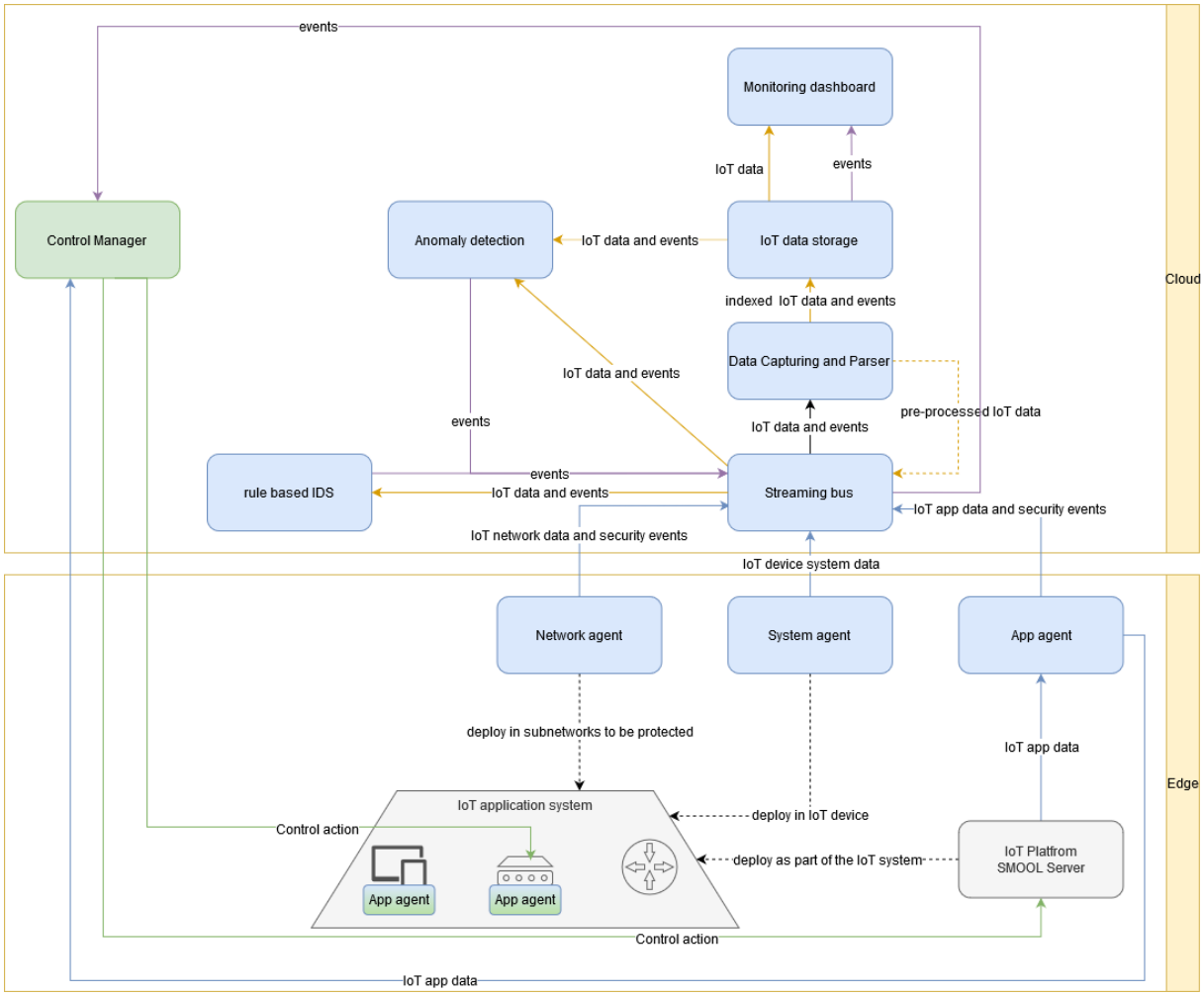

Figure 5. Architecture of ENACT Security & Privacy Monitoring and Control Enabler

4.1.1.3 Detailed design

The following sections explain more details of the Monitoring Enabler components. As introductory information, the Monitoring Enabler has been designed to be able to perform highly scalable monitoring and is based on Elastic stack provided technologies (i.e. Beats, Logstash, Elasticsearch and Kibana) [39].

4.1.1.3.1 Monitoring agents

The monitoring agents of the Monitoring Enabler are probes distributed in different components and layers of the SIS which are aimed at capturing different kind of data from the SIS, which can in turn be used to identify security events and detect security anomalies.

There are three types of monitoring agents that can be deployed in a SIS.

4.1.1.3.1.1 Network agent

Its objective is to acquire network traffic data of the SIS in order to identify security events. It is based on multiple sub-components explained in the following:

a. A N**etwork protocol analyser** based on t-shark [40] of Wireshark, the most used network traffic capturer and analyser.

b. ***Shipper for network data***, based on custom python scripts and Filebeat [41], which offers a lightweight way to forward and centralize the files with the acquired network data. It is crucial at this step to correctly define the mapping between the network attributes identified by t-shark and the indexes that are required to be created in the IoT data storage (see section 4.1.1.3.4). Even if this mapping can be done dynamically, there is a limit of maximum dynamically created indexes in Elasticsearch. And due to the huge amount of network attributes identified by the t-shark (since there are multiple protocols and network layers to capture and parse), defining properly the mapping is key.

c. ***Network Intrusion Detection System (NIDS)***: There are several well-known open source IDS tools available in the market as well as several sources of ruleset of known cyber security attacks. The Monitoring Enabler aims at enriching the security event monitoring by including one of these tools to the solution and tailoring it to the networks in the ENACT Use Cases.

   The Monitoring Enabler includes an open source signature-based IDS that is able to detect well-known security attacks. It is based on Suricata, which is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), network security monitoring (NSM) and offline pcap processing [43].

   Suricata identifies events based on pre-defined rules that are settled by connecting to open source and/or commercial database sources with knowledge of the famous cyber security attacks [44].

d. ***Asset discovery*** based on a network discovery tool in order to discover the active assets of the SIS (i.e., equipment and devices connected to the network). The implemented asset discovery service follows a hybrid approach; it combines active and passive asset discovery approaches in order to not overload the SIS network.

e. ***Anomaly detection***: The anomaly detection service on the Monitoring Enabler is executed at different levels: (i) at the edge level within the network agent, and (ii) at the Cloud level using all the data gathered from the SIS. At the moment, the anomaly detection service at the edge level generates security events related to ARP protocol (e.g. detecting ARP spoofing) and MQTT protocol (events based on rules for authorized users and assets).

4.1.1.3.1.2 System agent

Its objective is to acquire raw data regarding activities of users and processes of specific devices part of the SIS. It is based on Auditbeat [42] technology.

4.1.1.3.1.3 App agent

Its objective is to identify security events from the acquired operational data of the SIS and the data handled by the IoT platform. There are different kinds of app agents that can be used within the SIS; they operate at different levels of the SIS so they can be deployed and used independently in the SIS. There are two types of app agents based on the SMOOL IoT Platform, which are fully explained in Section 4.1.2 One of the agents, called Security KP, is continuously listening to all communications in and out through the SMOOL IoT Platform. Since the SMOOL KP is able to understand the semantics of the operational data used by the SIS, the app agent can acquire this data already parsed to the specific semantics of the SIS. It also uses Filebeat, which offers a lightweight way to forward and centralize the files containing the IoT operational data.

The second app agent, called security aware KP, is a customized application agent that is hosted in the IoT device and works with specifics of the application level. Therefore, different security aware KPs for the applications coexisting in the same IoT environment can be deployed, each of which would deal with the enforcement of the required security policy of a particular smart thing managed by the application.

4.1.1.3.2 Streaming bus

As mentioned in section 4.1.1.2, the streaming bus has a two-fold purpose. First, the bus allows to capture all the data sent by the monitoring agents in streaming and avoid a bottleneck at next phases of pre-processing of the IoT data. Second, it allows to manage all the data required by the components of the Monitoring Enabler and handle the publish-subscription among the Monitoring Enabler components. Therefore, the bus streams the pre-processed IoT data and events generated by different agents and components of the Security and Privacy Monitoring and Control Enabler.

It is based on Apache Kafka, an open source distributed streaming platform [45].

4.1.1.3.3 Data Capturing and Parser

The Data Capturing and Parser component allows to ingest data from heterogeneous sources simultaneously and parses and transforms it to facilitate the incident and anomaly detection. It is based on Logstash, which is an open source, server-side data processing pipeline that ingests, transforms and sends the data to other component(s) [46]. In the case of the Monitoring Enabler, it ingests the data from the streaming bus and after transforming it, it sends it to two other components: the streaming bus and the IoT data storage.

4.1.1.3.4 IoT data storage

The IoT data storage stores all the acquired and pre-processed data in a NoSQL database. It is based on Elasticsearch, a distributed, RESTful storage, search and analytics engine [47].

The IoT data storage defines multiple indexes based on the multiple sources of data (i.e. raw data from different monitoring agents and generated events by the IDS and/or control agents).

Following Figure 6 and Figure 7 show different indexes created by the acquired data, parsed and stored in the IoT data storage.

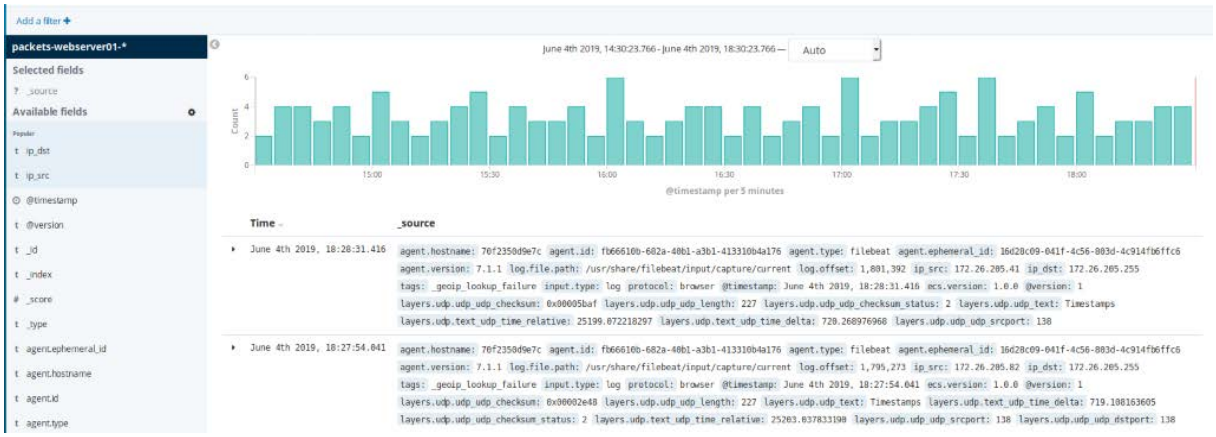

Figure 6. Example of indexes created in Elasticsearch

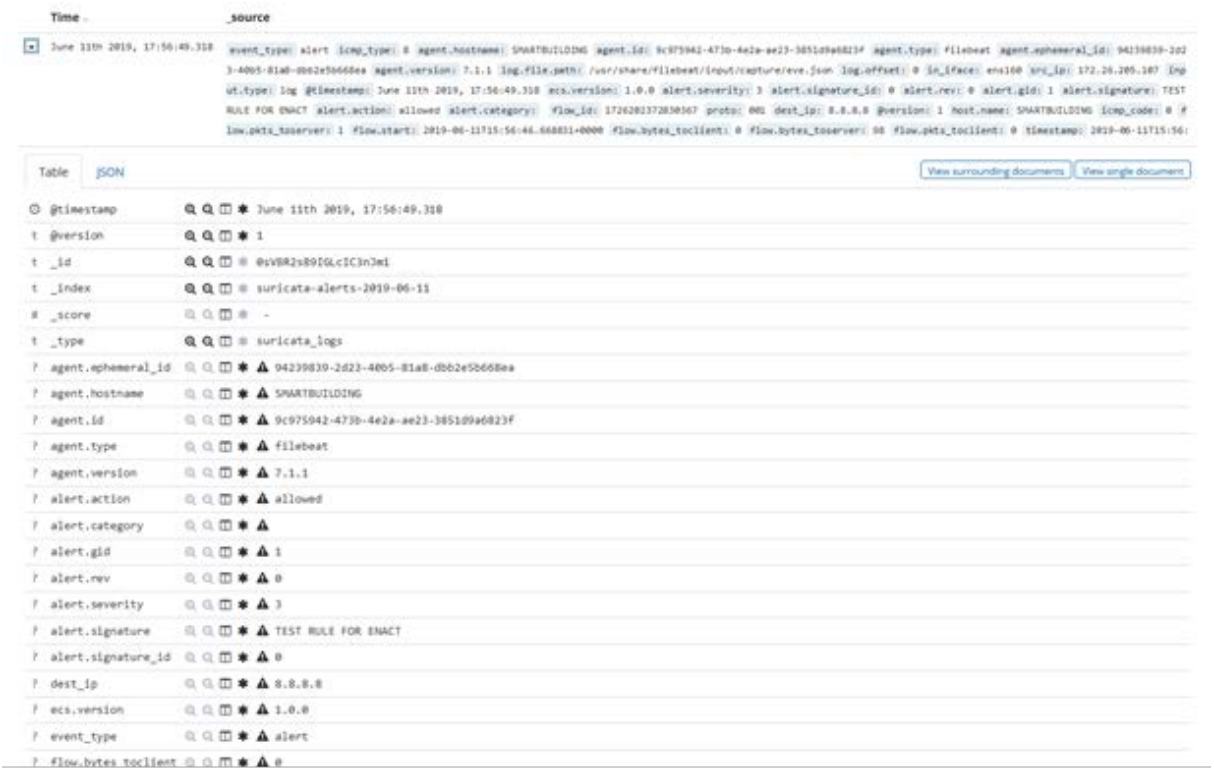

Figure 7. Example of indexes created in Elasticsearch

Next it is shown an entry in the database of a NIDS event, specifically an alert of “Scan Network”.


```
{
 "_index": "suricata-logs-2019-06-19",
 "_type": "suricata_logs",
 "_id": "XN15bmsB9IGLcIC3UR1T",
 "_version": 1,
 "_score": null,
 "_source": {
  "agent": {
   "hostname": "855db674acac",
   "id": "7c7a1b3a-c33b-4ee1-9bce-44248178af0a",
   "type": "filebeat",
   "ephemeral_id": "a61ad4e4-8255-4cc3-9efd-c46cba095d08",
   "version": "7.1.1"
  },
  "log": {
   "file": {
    "path": "/usr/share/filebeat/input/capture/eve.json"
   },
   "offset": 8621941
  },
  "tx_id": 0,
  "app_proto": "http",
  "src_ip": "172.26.252.100",
  "src_port": 60367,
  "in_iface": "enp2s0",
  "input": {
   "type": "log"
  },
  "@timestamp": "2019-06-19T06:42:51.860Z",
  "event_type": "alert",
```

```
  "ecs": {
    "version": "1.0.0"
  },
  "alert": {
    "severity": 3,
    "metadata": {
      "updated_at": [
        "2013_10_18"
      ],
      "created_at": [
        "2013_10_18"
      ]
    },
    "signature_id": 2017616,
    "rev": 4,
    "gid": 1,
    "signature": "ET SCAN NETWORK Incoming Masscan detected",
    "action": "allowed",
    "category": "Detection of a Network Scan"
  },
  "flow_id": 1942230808532336,
  "proto": "006",
  "dest_ip": "172.26.205.101",
  "@version": "1",
  "host": {
    "name": "855db674acac"
  },
  "http": {
    "protocol": "HTTP/1.0",
    "hostname": "172.26.205.101",
    "http_method": "GET",
    "length": 0,
    "xff": "104.152.52.23:44889",
    "url": "/",
    "http_user_agent": "masscan/1.0 (https://github.com/robertdavidgraham/masscan)"
  },
  "dest_port": 80,
  "flow": {
    "pkts_toserver": 4,
    "start": "2019-06-19T06:42:50.857456+0000",
    "bytes_toclient": 122,
    "bytes_toserver": 786,
    "pkts_toclient": 2
  },
  "timestamp": "2019-06-19T06:42:50.869138+0000"
},
"fields": {
  "flow.start": [
    "2019-06-19T06:42:50.857Z"
  ],
  "@timestamp": [
    "2019-06-19T06:42:51.860Z"
  ],
  "timestamp": [
    "2019-06-19T06:42:50.869Z"
  ]
},
"highlight": {
  "alert.signature": [
```

```
    "ET SCAN NETWORK Incoming @kibana-highlighted-field@Masscan@/kibana-highlighted-field@
detected"
   ]
  },
  "sort": [
   1560926571860
  ]
}
```


4.1.1.3.5 Monitoring dashboard

The Monitoring dashboard displays all acquired data and generated events in a user-friendly manner in form of alerts, statistics and graphs. It is based on Kibana [48]. The Monitoring Dashboard is composed of multiple viewpoints, which offers to the end user a complete overview of the SIS security status. Moreover, the Monitoring Dashboard provides ad-hoc adaptability in case the end user desires to customize the graphs and other visualization objects.

At the moment there are three different viewpoints in the Monitoring Dashboard:

i. General, which includes three types of dashboards: overview dashboard, anomalies related dashboard and finally a dashboard that shows all critical events lists.

ii. Network, which includes also three types of dashboards: network traffic dashboard, NIDS events dashboard and IoT Platform SOFIA-SMOOL dashboard. The last one also includes the events generated by the app agents.

iii. System, which includes two types of dashboards: overview of systems events and file integrity events dashboard.

Each of the dashboard offers the possibility of inspecting the data behind the graph or the visualization object.

Figure 8 shows the overview dashboard view of the General viewpoint in the Monitoring Enabler. It offers most important information related to the security events generated over the SIS to protect. The end user can navigate for the rest of the dashboard to learn about security events details. For instance, Figure 9 shows the network traffic dashboard of the Network viewpoint and Figure 10 shows the anomalies dashboard of the General viewpoint.

Figure 8. Overview view of the General viewpoint in the Monitoring dashboard

Figure 9. Network traffic view of the Network viewpoint in the Monitoring dashboard

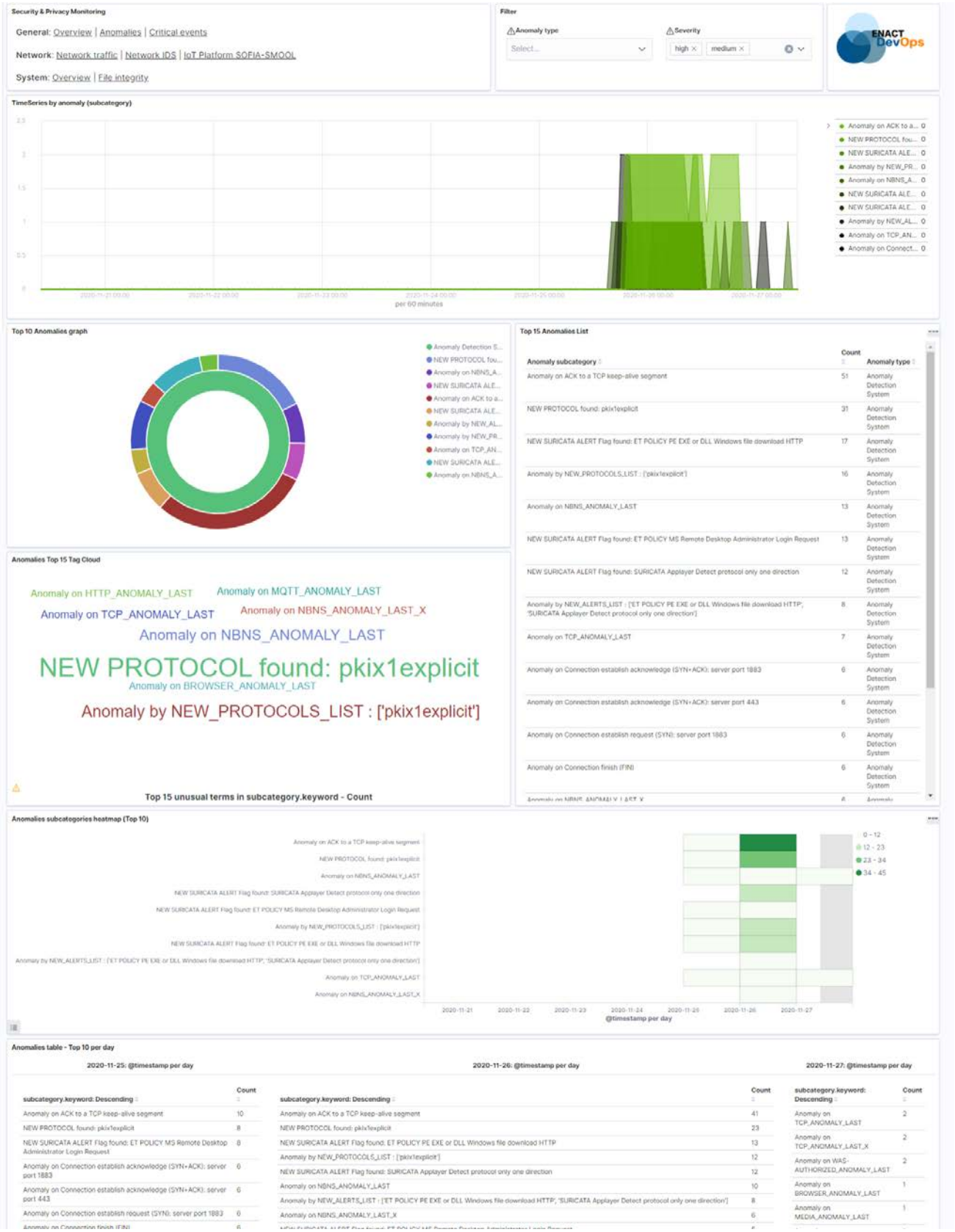

Figure 10. Anomalies view of the General viewpoint in the Monitoring dashboard

4.1.1.3.6 Anomaly detection

The Anomaly detection service of the Monitoring Enabler operates at different levels of the SIS. At the Edge level it is integrated into the network monitoring agent analysing specific network protocols.

At the Cloud level, the Anomaly detection service analyses all the IoT data and events of the SIS and applies ML techniques for the anomaly detection. Mainly, Deep Learning techniques such as Recurrent neural networks (RNNs) and Long short-term memory (LSTM) have been used to perform SIS behavioural analytics and anomaly detection.

Multiple ML and Deep Learning techniques have been researched in order to implement an efficient and accurate Anomaly based IDS. For instance, some of the LSTM techniques studied are: Vanilla LSTM, Stacked LSTM, Mixed LSTM, or Bidirectional LSTM. Figure 11 and Figure 12 show some results obtained for MQTT protocol predictions and detected anomalies with different LSTM techniques.

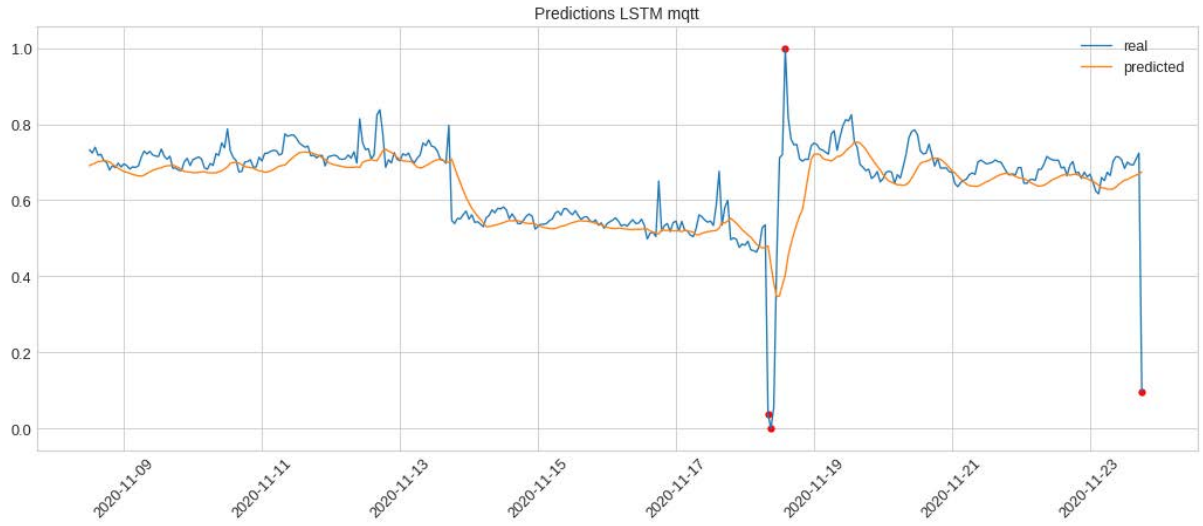

Figure 11. Vanilla LSTM predictions for MQTT

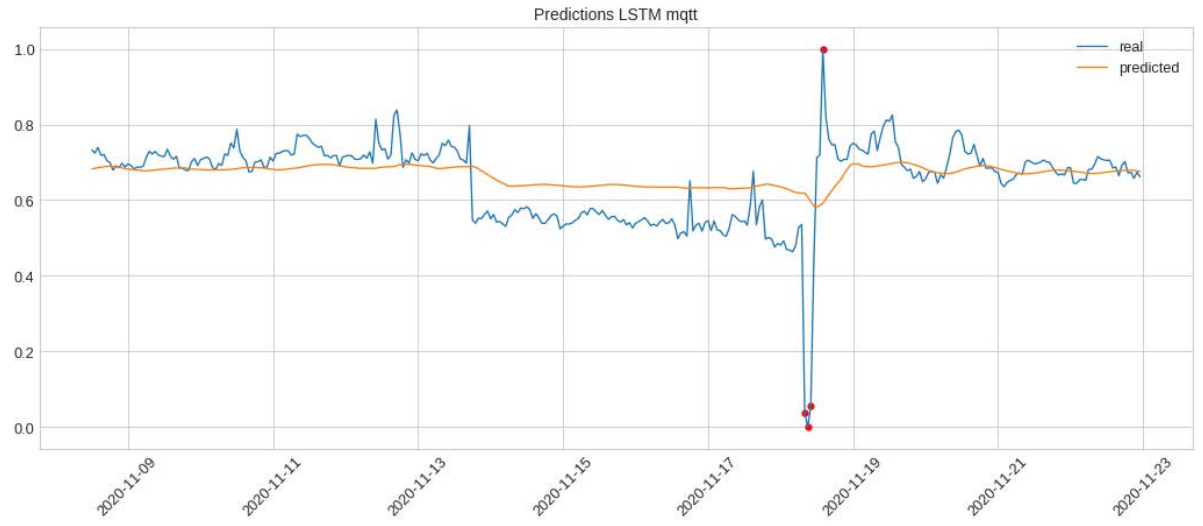

Figure 12. Bidirectional LSTM predictions for MQTT

The main types of anomaly events detected by the Anomaly detection service are listed next:

- Asset discovery related events
  - MAC change event
  - Discovery of new assets
- Anomaly based security system (Discovery of new and Anomaly detection)
  - Multiple network protocol anomalies
  - Network expert flag related anomalies
  - NIDS' alerts related anomalies
  - Authentication related anomalies
  - System events related anomalies
- Protocol specific related events
  - At the moment, rule-based security events detection for MQTT protocol is implemented.
  - ML based anomaly detection for MQTT published messages.

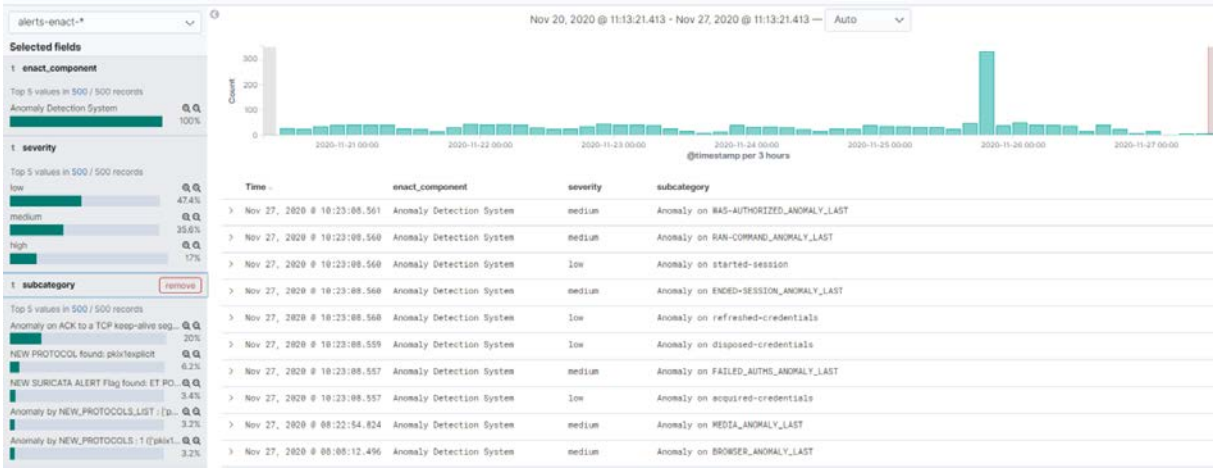

Figure 13. Snapshot of detected anomalies

4.1.1.4 Interface

As explained in section 4.1.1.3.5, the Monitoring Enabler provides a user graphical interface where DevOps teams can check the status of the security of the SIS. The Monitoring Enabler displays the information in form of statistical graphs and alerts. Moreover, the Monitoring enabler can provide all handled data internally to other ENACT enablers with a client for the streaming bus to subscribe to the events and data generated by the Monitoring Enabler [49].

4.1.1.5 Tutorial

The Monitoring Enabler is delivered as a docker based deployment. The main steps for its deployment are explained next.

Docker needs to be installed in all machines used for deploying the Monitoring Enabler components; Docker installation is well explained in the documentation of Docker [50].

The Docker images of the Monitoring Enabler components will be generated and published in an artefact repository managed by Tecnalia: https://artifact.tecnalia.com/. They are available for authorised internal use of ENACT partners.

Network monitoring agents

Instances of the network monitoring agents need to be deployed in each of the networks to be monitored in the SIS. In order to monitor the network traffic that is not being sent to or from the monitoring machine, the network interface of the machine needs to be enabled as promiscuous mode and on a switched Ethernet network, it also needs to be set up the use of port mirroring.

System monitoring agents

Instances of the system monitoring agents need to be deployed in each of the machines of the SIS to be monitored.

Streaming bus, Data Capturing and Parser, IoT data storage, Monitoring dashboard (Server-side components of the Monitoring Enabler)

The server component parts of the Monitoring Enabler need to be deployed in a machine that at least fulfils the following hardware requirements of the base technologies:

- Streaming bus based on Kafka: HW requirements https://kafka.apache.org/documentation/#hwandos
- IoT data storage based on Elasticsearch: HW requirements https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

App monitoring agent

The app monitoring agent is particular of SMOOL IoT platform and its source code is published in ENACT gitlab: https://gitlab.com/enact/smool_enact.

The installation and usage guides are included there in the README.md file.

This agent execution is closely related to the execution of the SMOOL Platform. The usage guide of SMOOL can be found here: https://bitbucket.org/jasonjxm/smool/wiki/Home

4.1.1.6 Main innovation

The main innovation brought by the monitoring service offered by the Security and Privacy Monitoring and Control Enabler can be summarised as follows:

- Flexibility to include all or only some of the components of the architecture, and to customise the design to particular needs of the use case in terms of e.g., security agents to be deployed, security policies and metrics to monitor, and tailored alarms and data visualisations.
- Holistic continuous monitoring that correlates data captured in the three main layers of the SIS: network, system and application layers.
- Richness of the analysis made which combines signature-based detection and artificial intelligence-based detection of incidents and anomalies.
- Rapid elasticity and full scalability of the tool for SIS that involve large amounts of sensors and actuators while it still produces an informed situational awareness of the overall system.

- Event bus for integration with other cyber security threat intelligence services supporting forensics and information sharing of cyber security incidents' indicators.
- Seamless integration with control mechanisms at application layer by relying on new developed SOFIA SMOOL capabilities in terms of secure communication monitoring and control agents' management.
- Advanced asset discovery feature included to keep continuous control of the SIS entities participating in the IoT environment.

The Security and Privacy Monitoring mechanisms have been adopted in one of the assets by Tecnalia and their innovative features are currently under publication process. A description of the data capturing layer in relation to the use of the mechanisms in an IIoT scenario for energy domain is available in [27].

4.1.2 Security and Privacy controls in IoT platform

As part of the security and privacy controls developed in ENACT, the Control Manager is able to manage the behaviour of the SMOOL IoT Platform (based on SOFIA technology [1]), which is the platform used in the Smart building Use Case in ENACT. This enables the control of the communications between the "things" in the IoT environment by profiting from the intermediary nature of the IoT Platform in such communications.

To this end, SMOOL has been extended with a number of features oriented towards the full control of security and privacy of the data exchanged in the environment. Particularly, the features developed in the final prototype to embed control capabilities in SMOOL are the following:

4.1.2.1 Security extensions to SOFIA middleware ontology

As described in deliverable D4.1, SOFIA is a semantic middleware that enables the communication between data publishers and subscribers on the basis of semantic concepts. The middleware relies in two main elements:

- *Knowledge Processors (KP)*, which are endpoints of the smart IoT applications where the logic of the IoT application is implemented. These endpoints produce and/or consume data to fulfil their tasks. An example of KP is a temperature sensor (of particular type and of particular vendor), a lightning actuator, a web service providing data, etc.
- *Semantic Information Broker (SIB)*, which manages the intermediation between the KPs. To this aim, it enables sharing ontology-based semantic information between KPs and acts as gateway that controls the communication technology/network (TCP/IP stack, Bluetooth, etc) of messages transmitted between KPs.

The SMOOL middleware is the open source version of SOFIA that is used in the Smart building Use Case of ENACT. This middleware has been enhanced in ENACT to be able to control some security and privacy capabilities of the KPs intercommunicating through SMOOL. Particularly, the ontology existing in the baseline SOFIA middleware has been extended to include semantic concepts capturing security metadata of the communications and capabilities of the "things" exchanging data (see Figure 14). These concepts include notions related to subscriber's and/or publisher's identity authentication, subscriber and/or publisher authorization, data confidentiality, data integrity and non-repudiation of the data origin and/or integrity.

The final version of SMOOL updates include the integration with ThingML, which is a software modelling language and tool set for the design and implementation of distributed reactive systems. ThingML is fully integrated with GeneSIS and ThingML-based components can be automatically deployed by GeneSIS.

The integration with ThingML enables that on IoT development projects where some of the autogenerated code is obtained by using ThingML files, SMOOL KPs, i.e. SMOOL clients, are specified in ThingML and deployed with GeneSIS deployment engine (see D2.3), which injects the security policy into the ThingML code of the client.

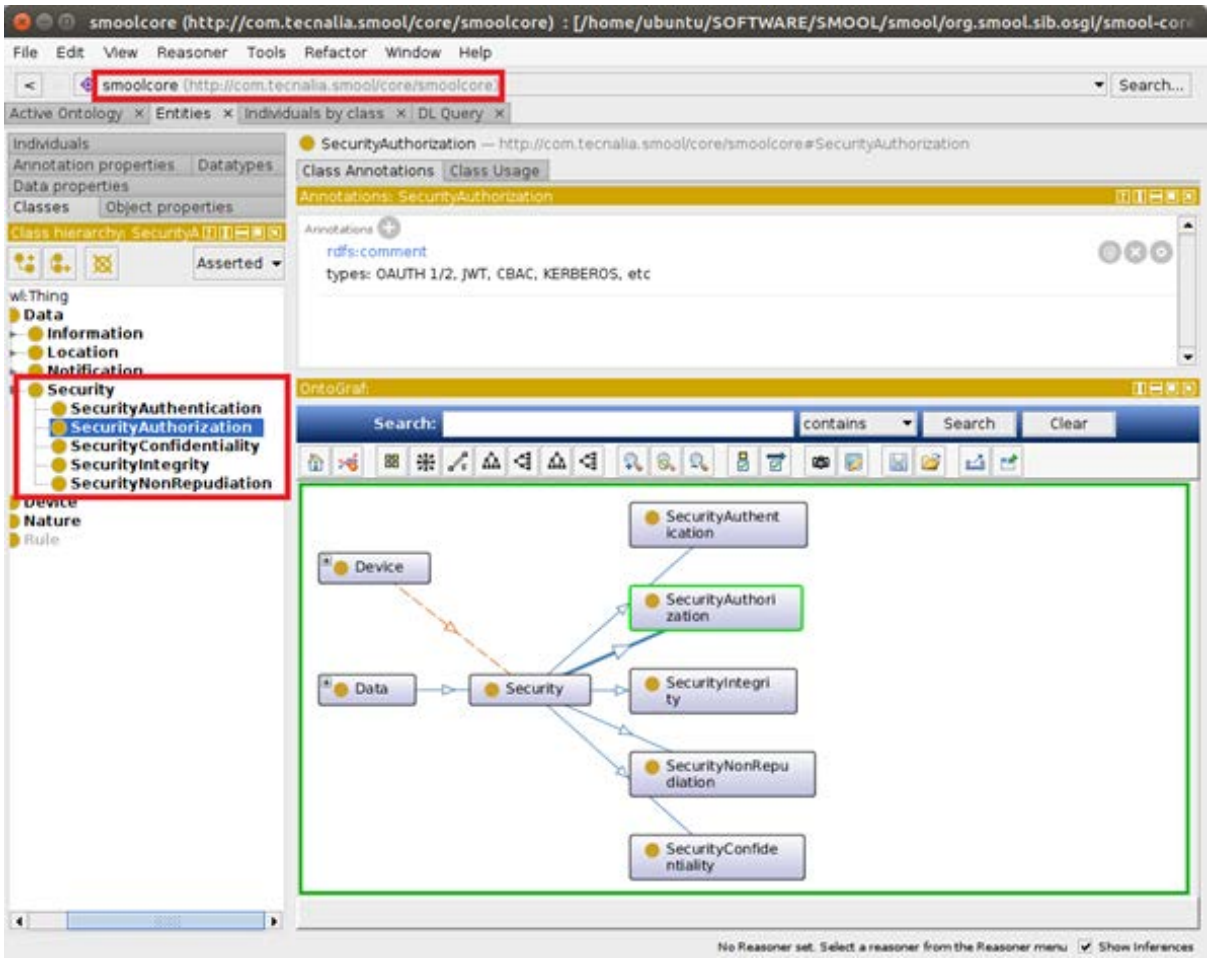

Figure 14. Security metadata extensions in SMOOL ontology

By means of these extensions in the ontology, the message format in SMOOL protocol has been enriched so as particular security attributes can be monitored in the communications. The SMOOL messages in Source Service Access Point (SSAP) interoperability protocol are transmitted in raw TCP (not Transport Layer Security -TLS- protocol), and therefore the security metadata could be used to implement within the middleware the security layer on top of raw TCP.

By using security properties as semantic data, SMOOL provides a mean to easily embed security monitoring and control to existing IoT environments (security-by-design). The DevOps teams can define SMOOL clients that leverage these security properties to check and enforce security concepts on messages requiring security controls. In the following sections we explain the details of how SMOOL clients provide such monitoring and control features and how they integrate with GeneSIS deployment.

In the following we provide the details of the monitoring and control methods offered through SMOOL platform and SMOOL clients.

4.1.2.2 Monitoring mechanisms in SMOOL

Within IoT environments or systems that use IoT platforms based on semantic middleware such as SOFIA, the communications between the things can be monitored by means of exploiting the communication gateway and interoperability features of the middleware. To this aim, SMOOL version of SOFIA has been extended in ENACT to create security aware clients of SMOOL which, besides being able to interpret SMOOL protocol messages, are able to perform security capabilities.

Two types of security aware SMOOL KPs have been developed, addressing a double layer of security in IoT environments:

a. **Monitoring at the overall IoT environment level:** A Security KP specialised in horizontal security for the IoT environment. The Security KP acts as an application agent (App agent in Figure 5) and logs all the needed information from the communications which are analysed by the Control Manager. Hence, the Control Manager acts as the security layer for the whole environment and some automatic detection rules are implemented in the Security KP to monitor the IoT environment security. Specifically, format and content of security metadata of the messages exchanged between the SMOOL clients are checked and alerts raised in case of non-conformities.

b. **Monitoring at individual IoT application level**: Security aware KPs can be used as customized application agents too (App agent in Figure 5) by IoT applications to check the desired security policy fulfilment in each smart thing (which has a SMOOL client associated). The design of the SMOOL client has been updated so it provides a placeholder for security policy monitoring and checking features, which could be programmed following one of these approaches: i) basic security policy checking of the default SMOOL client, or ii) more advanced policy checking features that can be programmed in the SMOOL client itself or, iii) an external security policy checking service that is invoked by SMOOL clients. This last option enables to update the security policy without the need to re-coding or even re-deploying the SMOOL clients.

This way, security metadata (e.g., tokens) sent in the communications between the KPs is gathered in SMOOL server logs and notifications of alerts sent by SMOOL can be seen in the Monitoring tool Dashboard. Figure 15 shows an example of SMOOL related security alarm information in the Monitoring dashboard. The number of alarms raised by SMOOL in the example is 25, there are 60 external connections attempts to the platform from the indicated source countries.

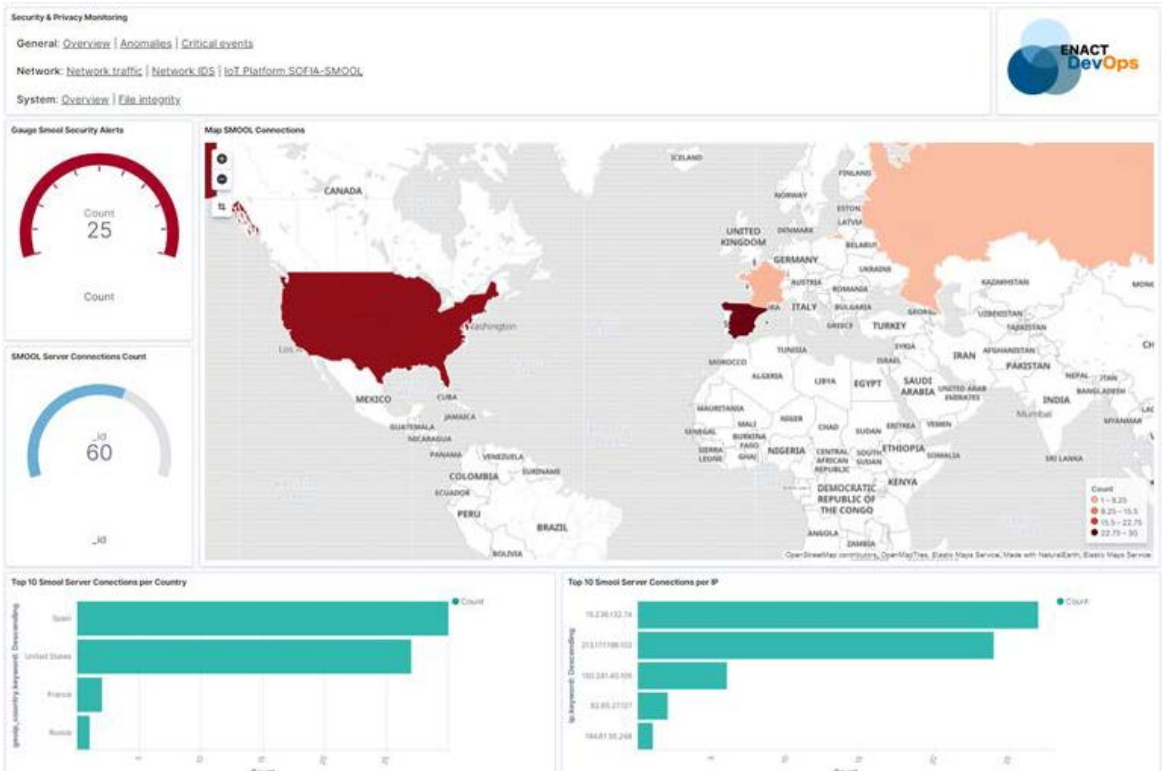

Figure 15. Example security alarms from SMOOL in the Monitoring tool Dashboard

4.1.2.3 Control mechanisms in SMOOL

Within the Security and Privacy Monitoring and Control Enabler architecture (Figure 5) an **app agent** has been included so as to control the application layer messages confidentiality and integrity. This app agent has been implemented as a special mechanism able to work with IoT systems that integrate SMOOL IoT Platform as the communication middleware between the things and/or between the things and the system services (potentially in the Cloud).

The app agents developed in ENACT for SMOOL-based IoT environments are of two forms:

a. **Control at overall IoT environment level:** A special SMOOL Knowledge Processor, the SMOOL Security KP, in charge of intercepting the communications through SMOOL for their further security analysis. The new Security KP acts as an application layer agent designed to collaborate with the Control Manager in the Security & Privacy Monitoring and Control Enabler. The Security KP is able to send application data and metadata to the Control Manager, which is the module taking the decisions and verifying whether specific detection rules apply over the data and metadata.

b. **Control at individual IoT application level**: The SMOOL KPs (i.e., SMOOL clients) include now a SecurityChecker class to enforce desired security policies. This enables that IoT applications using SMOOL KPs can define security policy checking and enforcing capabilities in each of the SMOOL KPs they use, which could be different for each of them, if necessary. Therefore, now SMOOL clients have a built-in security metadata checker to verify messages exchanged among SMOOL clients. In cases where a deeper control is needed, a specialised security metadata checker can be included in SMOOL clients, with additional privileges to watch and process the security metadata in messages exchanged, in the same way it is done with business logic concepts such as sensed temperature or gas values. This provides a fine-grained control on critical messages that may have significant security impact in the IoT system such as orders to actuators. More precisely, a client code can conduct security checks based on policies to be fulfilled by ontology concepts by using any of these options: (i) the default security metadata checker (for minimal configuration), (ii) a custom security metadata checker

implemented in the development phase (for full control of security), and (iii) a custom security metadata checker for integration with external security services.

Control at overall IoT environment level - Control through SMOOL Security KP:

The SMOOL Security KP is a full power SMOOL client that can request the Control Manager to check specific anomaly detection rules over the attributes of the IoT clients or the message itself. In case the anomaly detection rule verifies to true, the Control Manager would invoke the services in the SMOOL Server to control communications, i.e., request SMOOL Server to block the potential attacker IoT client or perform commands to start collecting additional data on the attacker prior to blocking it, etc. For example, the Control Manager can request that communications for a particular publisher or subscriber are blocked due to obsolete timestamps, unknown vendor, wrong encryption algorithm, obsolete certificate, etc.

The control mechanisms executed by SMOOL are complementary to other controls managed by the Control Manager in other layers, e.g., the context-aware access control by the CAAC tool in ENACT. While the controls in SMOOL are based on the analysis of application layer message content and they will only be available for IoT scenarios where SMOOL semantic platform is used, other controls will be adopted in other scenarios. For example, the CAAC tool will be used in scenarios where access policies to SIS services, resources or “things” will need to be adapted to context condition changes. In some cases, both SMOOL controls and CAAC could be used together as a double-layer protection mechanism.

Note that the communication between the Security KP and the Control Manager is secured by the use of HTTPS and an authorization header only known by both of them.

In the example message described in Figure 16 a SMOOL client (data producer) is sending encrypted metadata (encrypted with ChaCha20 algorithm) along with temperature information. The Security KP supports also the transmission of authorization data such as tokens (e.g., JSON Web Token (JWT), shown as commented code in green in the Figure 16). In the lower part of the figure, the SMOOL SSAP message is shown containing the security data transmission.

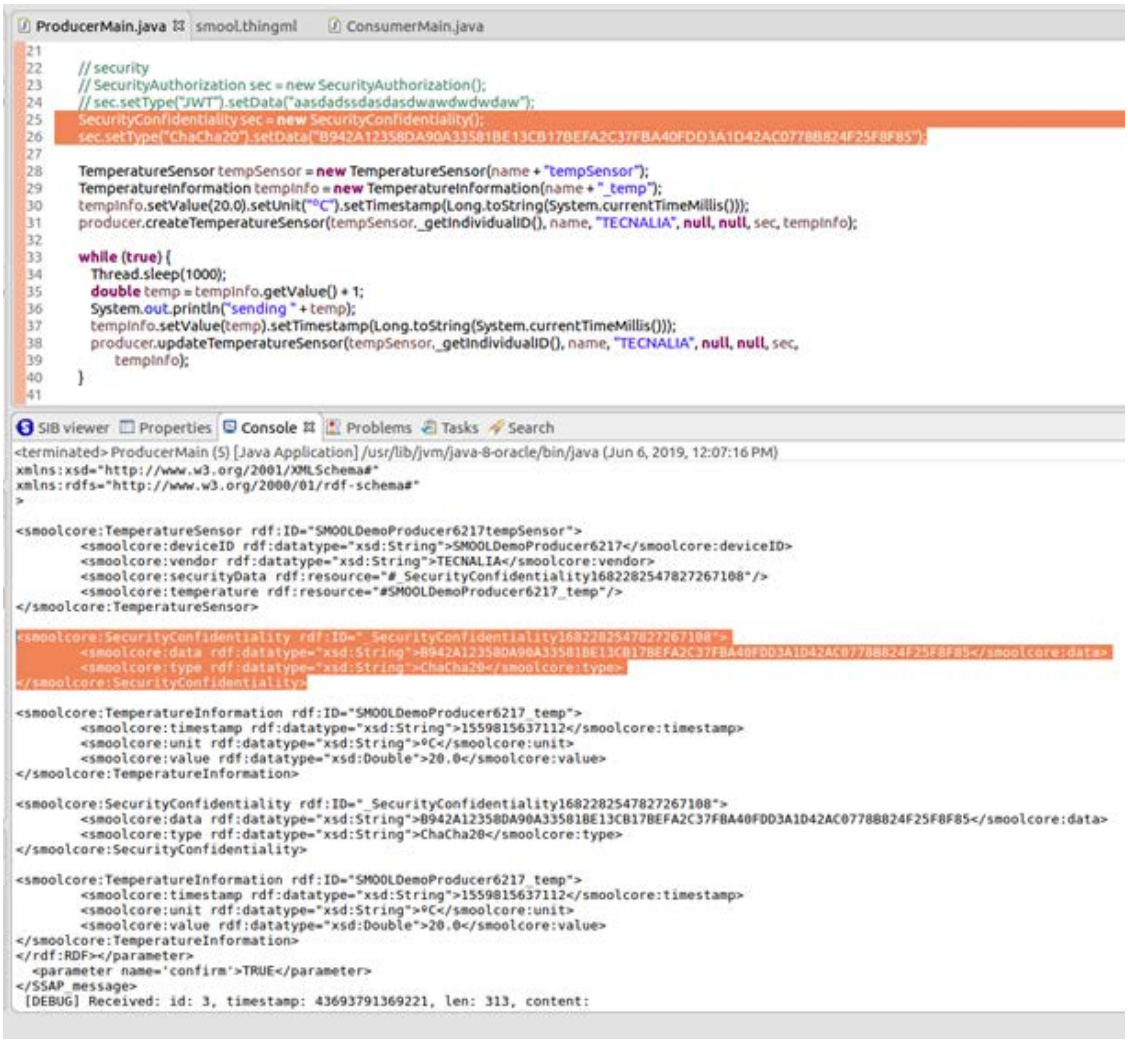

Figure 16. New message format in IoT Producers and Consumers to include security metadata

Control at individual IoT application level - Control through SecurityChecker in SMOOL KP:

SMOOL clients have been updated so that developers are relieved from manually specifying and maintaining security monitoring and control mechanisms in the code of the smart things. Instead, a developer can define its own SMOOL client, focusing on its business logic. Once the client is ready, s/he can specify how to deploy it also indicating the security and monitoring mechanisms that should apply to its SMOOL client. GeneSIS will then inject within the SMOOL client the necessary code to perform the security checks before actually deploying it.

In summary, through the collaboration between the developed SMOOL clients and GeneSIS the DevOps teams are able to reconfigure and update security mechanisms by design, in line with the evolution of IoT applications and the detected security and privacy threats.

To do so, we created a generic security component as a subtype of `InternalComponent` that represents a SMOOL client as a deployable artefact. This client can follow any of the security check options discussed above and is implemented with ThingML code, which integrates (i) the necessary SMOOL libraries and (ii) the SMOOL client business logic. The main rationale behind this choice is the following: ThingML offers an extra abstraction layer that provides the ability to wrap the code and dependencies that compose a SMOOL client and to inject into it the necessary security code. In addition, it provides GeneSIS with a standard and platform-independent procedure to generate, compile, configure and deploy the implementation of the security mechanisms. After the application is built and

deployed, this SMOOL client exposes a property `securityPolicy` that represents the security policy to be checked by the SMOOL security metadata checker.

The new implementation affects each KP in the following elements:

- ENACTProducer: this one retrieves the actuation orders sent by the IoT application. The current implementation will apply the "Authorization" policy automatically on that client. This is the main change. For the developer, the change is that the security control code has been removed from the current `Observer` object, because it is executed in the core implementation, so now the code in the `Observer` is cleaner.
- ENACTConsumer: no action, since this client is not subscribed to critical concepts (like actuation orders).
- ENACTSecurity: the `SecurityChecker` class has been tuned to bypass the messages because the business logic for this KP is related to security, so it is better to pass any messages, valid or invalid.

The new version handles most of the security automatically, by using an `org.smool.security.SecurityChecker` class. This class verifies that **any** message received fulfils the specified security policies. For example, according to the security policy in ENACT Smart Building Use Case, messages sent by IoT applications to actuators in the building should have valid authorization tokens, otherwise the message is rejected, i.e., the SMOOL client controlling the actuators would not process it and would not reach the target actuator.

This whole `SecurityChecker` (or `SecurityEnforcer`) could also be deployed from GeneSIS with more advanced features, like connecting to external security checking services or performing more specialized security constraints. The main achievements of the `SecurityChecker` are:

- The coding of basic security features that are offered in the smart things is transparent for KP developers, as it is now a built-in security in all SMOOL KPs. This facilitates the developers' task of embedding security in their smart things since they do not require to implement security code and can focus on the business logic. For advanced users, with greater security knowledge, the basic security code in a SMOOL KP can be tuned and more advanced security features can be added in this code, if needed.
- It provides a level of abstraction for third parties and no knowledge of SMOOL is required to implement custom security features on SMOOL KPs, because the `SecurityChecker` extracts all the security related semantic data for every concept. Therefore, external security features can be provided as simple description of policies (e.g., a JAR library as dependency), or custom `SecurityChecker` class with connection to external security policy checking services.
- By default, if no `SecurityChecker` is found, or no security policies have been set, then no action is performed (while keeping compatibility with previous versions of SMOOL KP Producer, this allows customized security per KP).

Example of adding a policy in the `SecurityChecker` for a java based KP:

```
public SecurityChecker() {
        policies.put("BlindPositionActuator", "Authorization");
}
```

In the simple policy above the `SecurityChecker` would check if all SMOOL messages targeting `BlindPositionActuators` have a valid "authorization" token in the `SecurityAuthorization` field of the message.

The `SecurityChecker` verifies the policy fulfilment at the time when messages of semantic topics the KP is subscribed to arrive at the KP, and it works as follows:

- The SMOOL KP tries to load the class `org.smool.security.SecurityChecker`. If not found, no further action is performed.
- If found, the default `org.smool.security.SecurityChecker` class checks every message ARRIVED to the SMOOL KP and returns either `true` if no security policy is applied or the policy is fulfilled, or `false` if the policy is not fulfilled.
- All the SMOOL auto-generated subscription class will check the binary response of the `SecurityChecker`. If false, the subscription will not emit the subscription event. That is, the message is STOPPED in the subscription class, so the IoT apps are safe from malicious messages arriving.
- The IoT app including SMOOL KPs will not notice any incoming message, since subscription observers are not triggered. Only valid messages will arrive to the customized `Observer` objects.
- External third parties can use the basic `SecurityChecker` as template and replace it with their own security checking implementation.

4.1.2.4 Tutorial

The detailed description of how to use SMOOL monitoring and control mechanisms and how to create SMOOL KPs with security can be accessed here: https://bitbucket.org/jasonjxm/smool/wiki/Home.

4.1.2.5 Main innovation

The main innovations brought by the security extended SMOOL IoT platform and collaborating SMOOL KPs are outlined below:

- The enhanced SMOOL offers an open source security ontology over SOFIA middleware to enable the comprehensive monitoring and control of a) all communications between smart things and IoT Platform, and b) communications between smart things in the IoT environment.
- The new SMOOL offers a good separation between the business logic and the security implementation in the SMOOL clients. The clients now include a security checker dealing with security policy enforcement, which can be tailored to the IoT system needs. The checker can support basic policy checking, extended policies or even invoke other external security services.
- The monitoring and control of secure communications in terms of keeping confidentiality, integrity and availability of data transmitted can be done either at whole IoT environment level, at individual application level, or at smart thing level, according to the needs of the use case. That is, the solution is able to combine, monitor and enforce different security policies that may apply at different layers of the SIS.
- The solution is fully integrated with GeneSIS deployer (which deploys server-clients of SMOOL) and other external security services (such as access control enforcement) which may be deployed by GeneSIS too, which eases the automation of the enactment of security aware SIS models.
- The solution is also fully integrated with ThingML which relieves DevOps teams from security coding burden since they already have a basic built-in security in the SMOOL clients. Furthermore, the basic security can be easily enhanced by just adding more features to control in the security policy to be enforced by SMOOL clients' security checker.

The results of the research on SMOOL enhancements have been preliminary described in the following international publications:

- Gallon, A., Rios, E., Iturbe, E., Song, H. and Ferry, N. Making the Internet of Things More Reliable Thanks to Dynamic Access Control. Book Chapter in Security and Privacy in the Internet of Things: Challenges and Solutions, IOS Press, 2020. ISBN: 978-1-64368-052-1, DOI: https://doi.org/10.3233/AISE200005

- Ferry, N., Nguyen, P., Song, H., Rios, E., Iturbe, E., Rego Fernandez, A., Martinez, S., Continuous Deployment of Trustworthy Smart IoT Systems. In Journal of Object Technology (JOT), AITO, 2020.
- Ferry, N., Dominiak, J., Gallon, A., González, E., Iturbe, E., Lavirotte, S., Martinez, S., Metzger, A., Muntés-Mulero, V., Nguyen, P.H. and Palm, A., 2019, May. Development and Operation of Trustworthy Smart IoT Systems: The ENACT Framework. In International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (pp. 121-138). Springer, Cham.

4.2 Context-Aware Access Control[14]

4.2.1 Purpose

In general, IoT systems link many devices such as sensors, cameras or smartphones to the Internet. These devices have the capacity to act as sensor or actuator in their environment, in a continuously changing context. Environmental data are considered as dynamic and give crucial information about a context (state of devices, user's behaviour and location, etc.). The traditional mechanisms of access control do not use these contextual data while making authorization decisions.

The objective of the Context-aware Access Control (CAAC) is to provide mechanisms for controlling the security, privacy and reliability of smart IoT systems. A specific emphasis is made on confidentiality by ensuring proper access control to the data, and on privacy, since the control over the personal data is kept by their owner. These mechanisms include functions to adapt the access control of connected objects according to the context of the application, in order to deliver context-based dynamic authorization, applicable to both IT and OT (operational technologies) domains.

Evidian Web Access Manager (WAM) provides security features for identity management and access control based on the OAuth2 [24] and OpenID Connect (OIDC) [25] protocols. The **Context-Aware Access Control tool** is an evolution of the authentication and authorization mechanisms provided by WAM intended for the Internet of Things.

As described in [26], context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

In the IoT use cases, context of devices and users is continuously changing. To make the security mechanisms more applicable to IoT use cases, they must be adjusted by using the contextual information.

In this objective, we extended the protocol OpenID Connect to also consider contextual data as dynamic attributes which can be taken into account when applying security rules.

4.2.2 Architecture

The Context-aware Access Control tool provides an Authorization mechanism that issues access tokens to the connected objects after successfully authenticating their owner and obtaining authorization. This authorization mechanism may be coupled with contextual information to adapt the access authorizations according to them (for example to make certain information more widely available in some urgent case).

To this objective, the Access Control Tool directly communicates with a Risk Server to make dynamic access controls based on the context information during the authorization phase. For example, it can reject the authorization if the access token is valid while other context information does not respect the authorization policy.

[14] EVIDIAN Proprietary software. The Interface documentation is available, and the API can be exposed for testing and validation purpose.

The authorization policy is a set of rules that define whether a user or device must be permitted or denied accessing to a backend server. An administrator can control this adjustment and create special authorization rules based on the context data provided.

The Context-aware Access Control tool is provided inside an infrastructure aimed to gather contextual information to deduce a risk level associated with a user. Figure 17 gives a global view of this infrastructure.

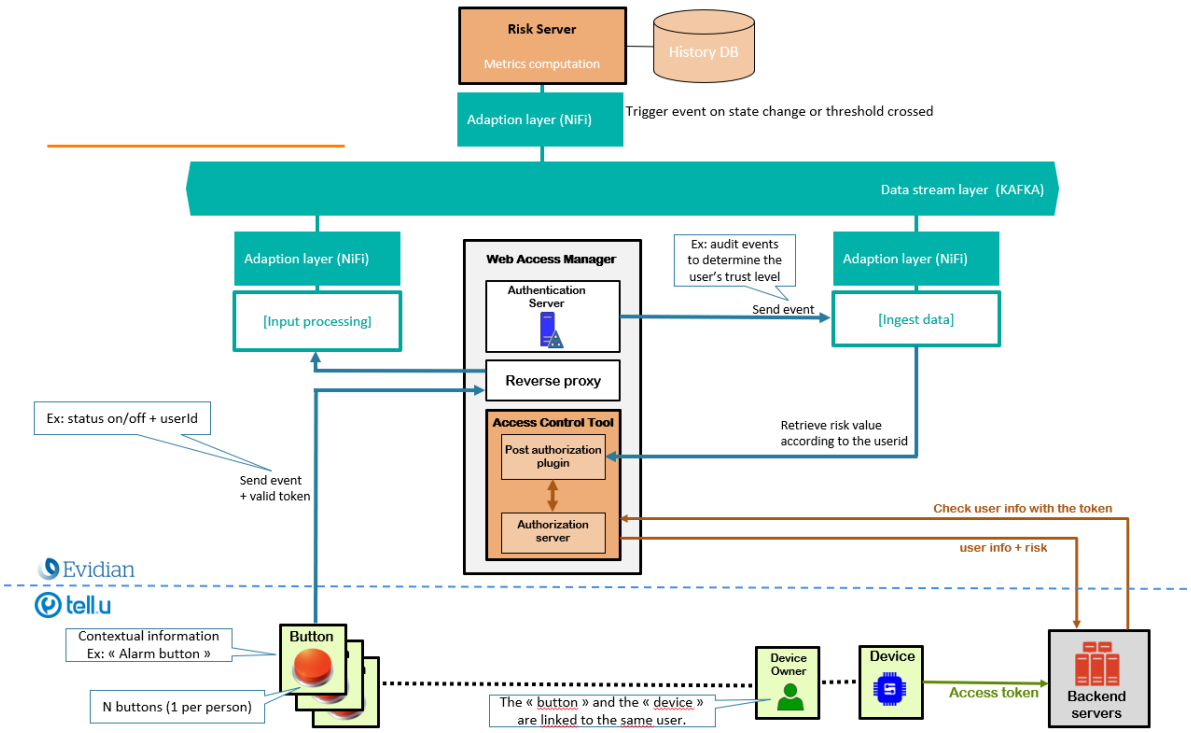

Figure 17. Context-aware Access Control global infrastructure

The infrastructure is based on the Apache Kafka event streaming platform [45], which allows to publish and subscribe to streams of events. The principle is to publish in this platform contextual information which may come from connected devices (sensors, alarms, etc.) or from audit events produced by the Evidian user access control solution. The contextual information is sent to an input processing interface which then publishes it to a Kafka topic. An event is then received by the risk server from an Apache NiFi interface [57] which will take into account the contextual information in a dynamic risk level computation. Then, when a device tries to access a resource, the CAAC retrieves the dynamic risk value associated to the device owner, and this is transmitted to the back-end server to modulate the access accordingly.

In this architecture, two components are providing the Context-aware Access Control mechanisms: The Access Control Tool and a Risk Server.

4.2.2.1 The Access Control Tool

The Access Control Tool of WAM depicted in Figure 18 is composed of an Authorization server associated to a Post authorization plugin, to add more controls during the authorization phase. Its purpose is to check if the request is authenticated and is authorized to access the backend server.

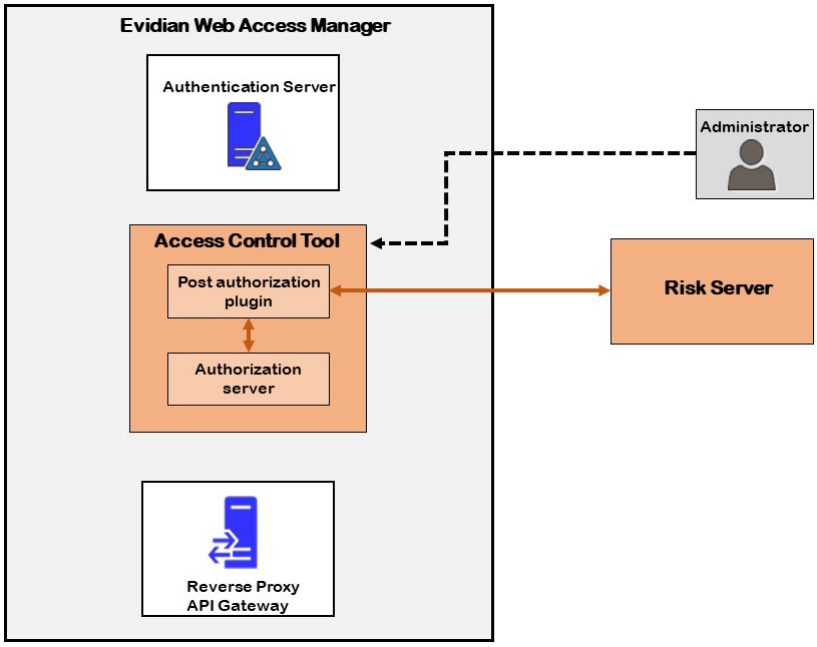

Figure 18. Access Control tool

Indeed, each time a device sends a request to a backend server, WAM can check the dynamic claims and scopes consented by the user associated to the device that performs the request and, in turn, realize special actions according to this information such as blocking the request or limiting the accessible resources. This is detailed in section 4.2.3.1.

The Post authorization plugin extends the basic authorization phase and is entirely customizable. Any operation can be executed during the authorization phase, including calling external programs, and in particular the Risk Server. The Post authorization plugin can create injection variables that can be reused and injected in the initial request sent to the backend server. This is detailed in section 4.2.4.

The Administrator can configure the Access Control Tool to adjust the authorization security rules according to the dynamic attributes.

4.2.2.2 The Risk Server

The Risk Server essentially relies on WAM audit events to calculate a risk level for each user. This allows detection of abnormal behaviour such as connections from unknown IP addresses, or multiple failed connections.

Contextual information coming from external sources (sensors, other applications, etc.) makes it possible to modulate this risk, i.e., to increase or decrease it depending on the situation. For example, in the case of a fire, a smoke detector immediately sends information to the Risk Server, which will considerably reduce the user's risk and allow easier access to resources.

The contextual information is dynamic and changes periodically.

During the authorization process of a connected device, if WAM is asked for the device owner’s information, it adds to the user’s attributes a user’s contextual data which is the dynamic risk level computed by the Risk Server. WAM links in its repository the user’s profile and the user’s dynamic information (the risk level). According to this risk value, the request of the device to access the backend server can be rejected if this value reaches a critical threshold. This is detailed in section 4.2.4.

4.2.3 Detailed design

4.2.3.1 Authorization mechanisms

The Authorization mechanism provided by the Context-aware Access Control tool uses the OAuth2 protocol, which provides authorization delegation mechanism. Following this protocol, an object can access a backend API by using an access token containing the list of claims (i.e., user's attributes such as username, email address, etc.) and scopes (read-only, read/write) that an authenticated user has consented for this object to access. An access token contains an authentication proof and the list of consented scopes and claims to access the asked resource.

The Context-aware Access Control tool provides devices with access tokens. The scopes and claims contained in the access tokens are used to restrict accesses to the backend server APIs to a consented set of resources.

The Authorization mechanism could be coupled to a multi-level, multi-factor Authentication Server that provides strong authentication mechanisms to the users. This mechanism mitigates the level of authentication required depending on the user's environment context and an external context. The risk is a value computed either statically, depending on a defined configuration, or dynamically by using a REST API to dialog with an external decision engine. The input used to compute the risk is the user's session context, which contains the browser DNA, the service the user wants to access and the configured trust level of this service, the access time and the trust planning associated to the service, and the IP address and its geolocation. Depending on the evaluated risk of the user's session, the level of the required authentication will be leveled up, or, if the risk is too high, the connection will be refused.

During the enrolment phase, a connected object is associated with a user; then the connected object can push data to a backend server in a controlled way by using the access token it received from the Access Control Tool. The backend server stores this data and can display it within an application. The authorized users and applications can retrieve the data from the backend server. WAM plays a pivotal role between all these exchanges by making authorization decisions depending on the context.

The Context-aware Access Control tool built on the OpenID Connect protocol includes a scope system. The scopes are used by an application to authorize access to a user information, like name and email. In the OpenID Connect protocol, the scopes system only returns a set of static user attributes. The Context-aware Access Control tool extends the protocol to also consider dynamic attributes which provides a contextual risk level on the user. The risk level is evaluated for a user, depending on his behaviour and on context information that may come from the devices he owns (see section 4.2.3.4). That way, the access control can be adapted depending on the context which can be continuously evolving, in order to make the access rights more secure and efficient in function of the current environment.

4.2.3.2 User's consent

During the user's device code flow interaction, the user gives his consent through the interface shown in Figure 19. The deviceID and/or the SerialNumber provided by the device are displayed. The user may enter an additional user defined DeviceID and/or a user defined SerialNumber.

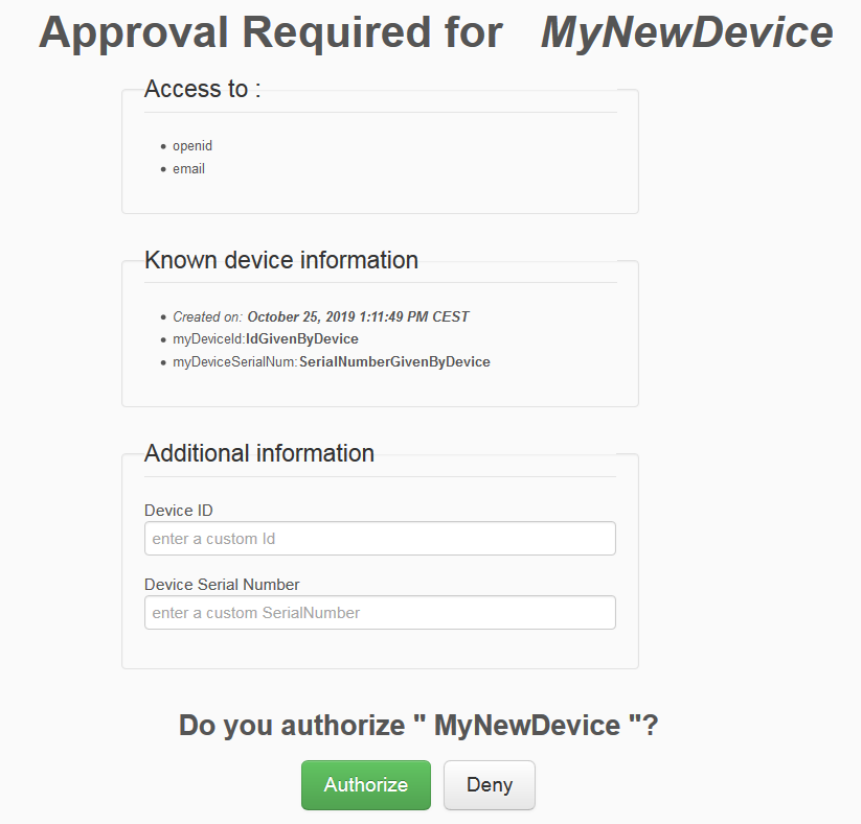

Figure 19. User's consent interface

At any time, the user can access through the WAM portal the "OpenID Connect provider access management" service shown in Figure 20, from where he can manage the tokens associated to his devices:

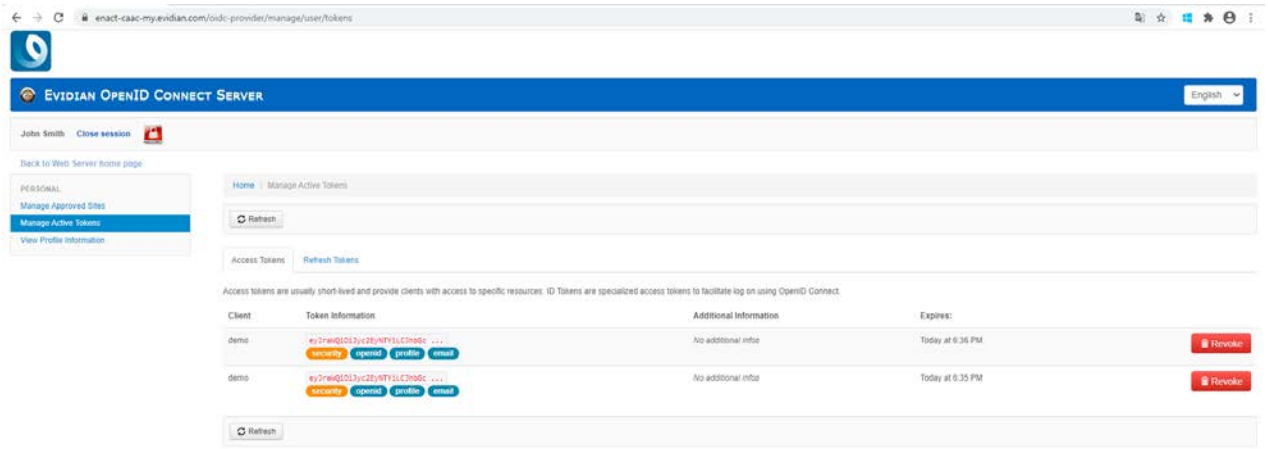

Figure 20. OpenID Connect provider access management interface

From this interface, the user can at any time revoke the granted tokens.

4.2.3.3 Managing contextual information

Contextual information which may come from connected devices (sensors, alarms, etc.), can be managed by the Risk Server to evaluate the risk level associated with a user, in addition to the audit events produced by the user access control solution.

The contextual information is sent to an input processing interface which then publishes it to a KAFKA topic.

In order to control this contextual information transmission, the transmitting device must be enrolled, and it must have received a valid access token which allows it getting its owner's userid. A simple test interface is provided to send a sample contextual information from an alarm button (on/off) to the NiFi interface. This is done with the following parameters:

```
userid
      REQUIRED. the "sub" field from the userinfo data.

button
      REQUIRED. the button status: "on" or "off".
```

This interface is available on https://enact-caac-context-input.evidian.com/contentListener. It can be tested by using the tutorial platform described in section 4.2.5.

4.2.3.4 Risk computation

A simple risk server is provided, which is intended to validate the architecture and to provide risk levels depending on both the contextual events sent by external devices and the audit events generated by WAM following the users' connections. This Risk server is a lab prototype based on the Quarkus framework [58], which will be further improved to become a component of a new future module "Evidian Prescriptive IAM". The current simple prototype is used in ENACT to demonstrate the functioning of the CAAC tool and its integration into the future Cloud architecture of the "Prescriptive IAM" module.

User's risk level:

The Risk Server essentially relies on WAM audit events to calculate a risk level for each user. This allows detection of abnormal behaviour such as connections from unknown IP addresses, or multiple failed connections.

A user's risk level is the level of trust placed in the user. On it depends the level of trust that can be placed in the devices that belong to him.

The user's risk level is based on a system of sanctions/rewards depending on the user's behaviour. Its computation uses a ranking system based on a user-specific score: the Risk Score.

A user's action represents an operation that the user (or his devices) does through Evidian WAM, for example: authenticate, log out, change password, etc.

We have chosen to divide the user actions into three distinct categories:

- Penalizing actions: successful connection from unknown IP addresses, multiple connection failures.
- Rewarding actions: successful authentication (with known IP address).
- Actions with no effect: disconnection or other.

Each action is linked to an audit event sent by WAM (specified by the EventType and the EventCategory of the event) and is associated to a weight, since the reward or penalty linked to the action is different

according to its importance and seriousness. After each new action, the value of this weight is added to the user's risk score.

In addition, an audit event history system allows to detect if an IP address is unknown or can be used by a user. This history is specific to each user. The notion of "IP address risk level" has been defined to estimate the risk associated with an IP address, depending on the number of occurrences in the audit event history.

Contextual information:
Contextual information coming from external sources (sensors, other applications, etc.) is used to lower the level of risk associated with users (or devices).

Each device providing contextual information must be enrolled in WAM and associated with a user. It is associated with a risk factor which can be used to modulate the user's risk score.

This modulation is done as follows: (Contextual device factor) * (User's risk score).

Then the risk level is deduced again from the new Risk Score.

4.2.4 Integrating the Context-aware Access Control tool

4.2.4.1 Device enrolment

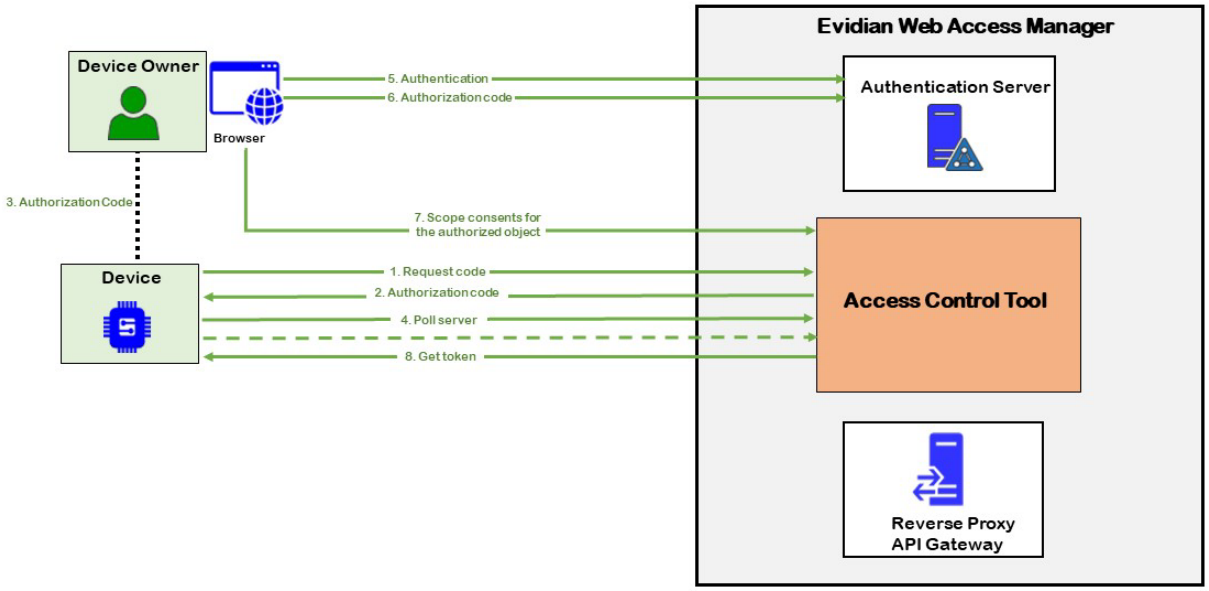

Figure 21. Device enrolment

The device enrolment procedure depicted in Figure 21 allows a device to be associated with the identity of its owner. The Access Control tool leverages on the OAuth 2.0 Device Flow protocol to achieve this.

The only requirements to use this flow are that the device is connected to the Internet and able to make outbound HTTPS requests, and that it is able to display or otherwise communicate a URI and code sequence to the user, and that the user (device owner) has a secondary device (e.g., personal computer or smartphone) from which to process the request. There is no requirement for two-way communication between the OAuth client (i.e., the connected device) and the end user's user-agent, enabling a broad range of use-cases.

During this procedure, the user gives his consent to the device to access data scopes on static attributes (username, email, etc.) and also dynamic attribute (a risk level computed from contextual information on the user). At the end of the enrolment phase, the device receives an access token. The device has now access to the device owner profile that includes static attributes (username, email, etc.) but also dynamic risk level.

The sequence diagram for this device enrolment procedure is described in Figure 22.

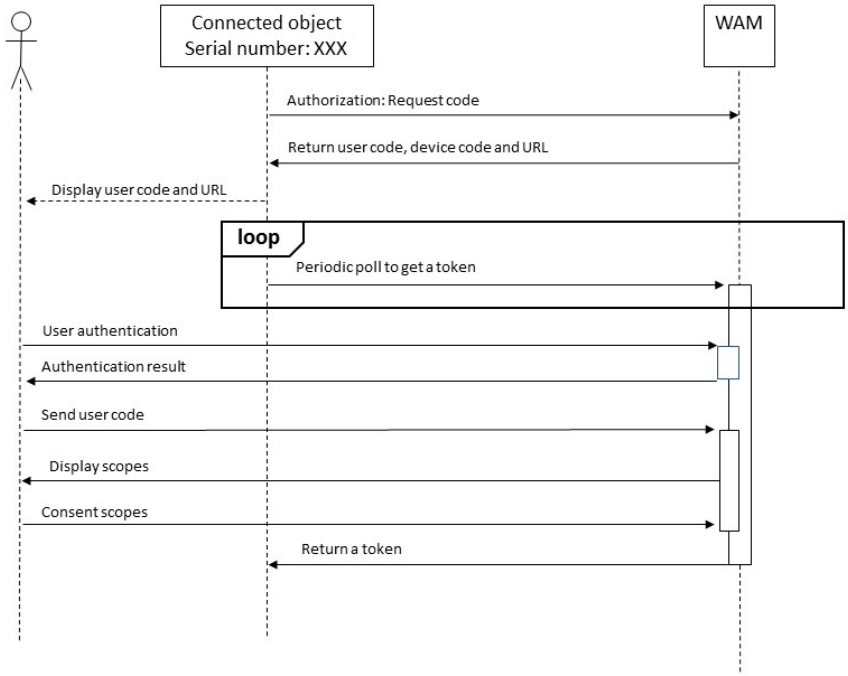

Figure 22. Device enrolment sequence diagram

During this procedure, the user gives his consent to the device to access data scopes on static attributes (username, email, etc.) and also dynamic attribute (a risk level computed from contextual information on the user). At the end of the enrolment phase, the device receives an access token. The device has now access to the device owner profile that includes the static attributes but also the dynamic risk level.

The Context-aware Access Control tool can work in various ways when integrated in use cases:

4.2.4.2 Context-aware Access Control with WAM used as reverse-proxy

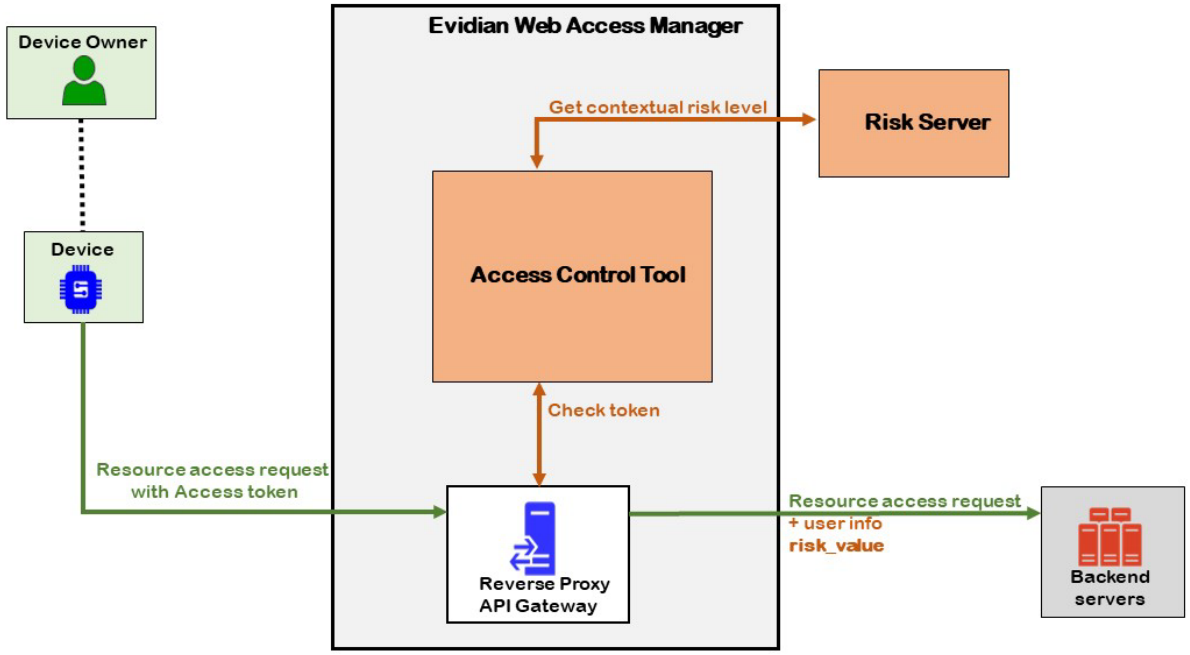

Figure 23. Context-aware Access control with WAM as reverse-proxy

In this case depicted in Figure 23, WAM is used as a Reverse proxy to protect the backend servers. Additionally, WAM checks the token of the incoming request to verify if the device is authorized to access the backend server. If this is the case, WAM injects in the header of the initial request the consent scopes of the device owner. This injection does not modify the request and the scopes injected contain some information about the device owner (username, email, etc..) and a risk level computed from contextual information. This allows the backend server to make the link between the requesting device and the user associated with it, and to know the risk level depending on the context.

Let's take an example of a connected arm tensiometer that needs to push some tension data to a backend server. Here the role of the backend server is to store the data received from the tensiometers. To do this, the device will make an HTTPS POST request with its access token to the reverse proxy that protects the backend server. WAM checks the validity of the token. Then during this authorization step, it gets the contextual risk level provided by the Risk Server. For instance, the risk level may be computed from contextual dynamic information about the position of the user (standing or lying down), or the temperature of the room, while the arm tensiometer is active. This allows to consider the data provided by the arm tensiometer in the context of the patient's environment. If the token is valid, WAM injects the consent scopes in the header of the original request. This consent scopes injected can be static like the identity of the user (username, email) or dynamic like the risk value related to this specific user. The backend server can now store the tension data associated to the user and the related contextual risk level.

The sequence diagram for this working mode is described in Figure 24.

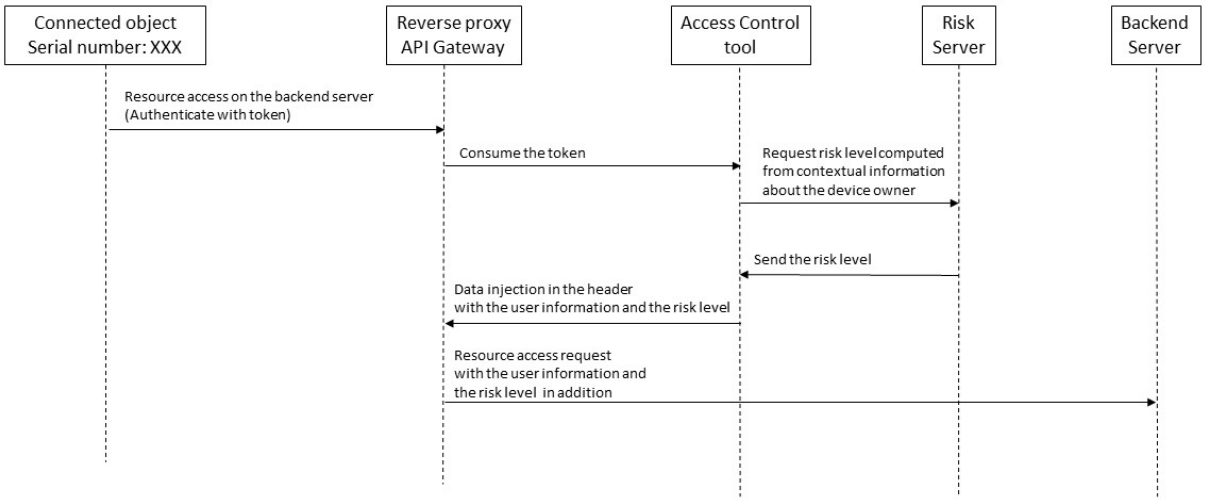

Figure 24. Context-aware Access control with WAM as reverse-proxy – Sequence diagram

This sequence diagram explains the mechanism of Context-aware Access Control with WAM acting as a reverse-proxy. Each time a device performs a request to a backend server with its access token, WAM consumes the token to verify the identity of the user associated to this device. After this, WAM retrieves the contextual risk level about the device and the user from the Risk Server. Then these data are injected in the header of the initial request and the request is sent to the protected backend server.

4.2.4.3 Context-aware Access Control with WAM used as OpenID Connect IDP

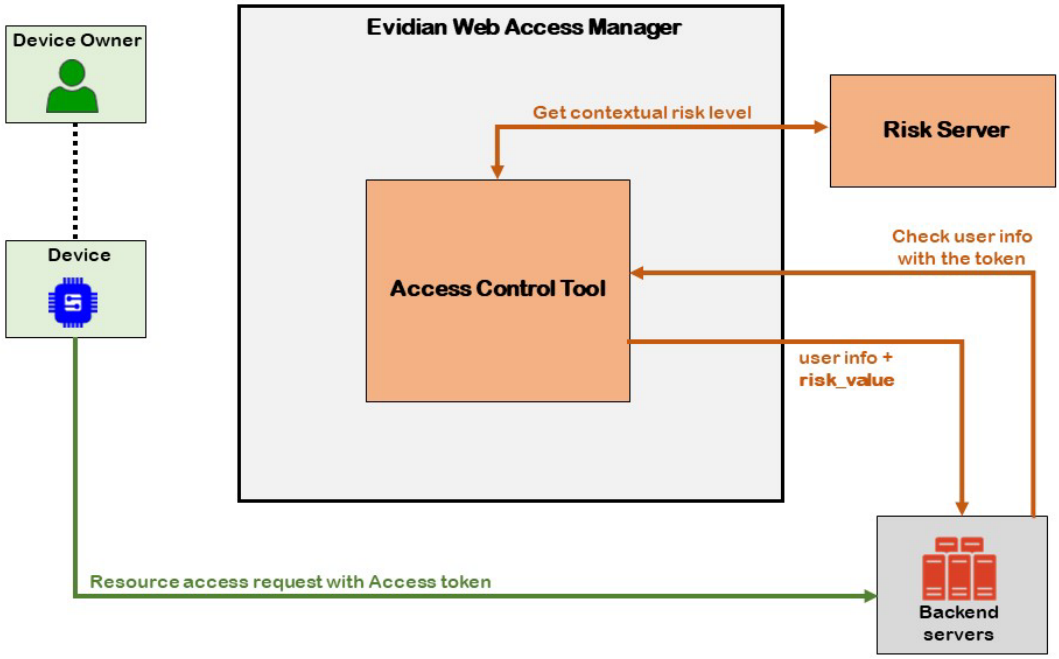

Figure 25. Context-aware Access control with WAM used as OpenID Connect IDP

In this case depicted in Figure 25, WAM is used as an OpenID Connect IDP. WAM only handles the access control part by checking if the token sent to a backend server is valid. If this is the case, WAM responds with the consent scopes of the device owner. These scopes contain some information about the user (username, email, etc.) and the contextual risk level.

Taking our previous example, the connected arm tensiometer needs to push data to the backend server. To do this, it directly makes an HTTPS POST request with its access token to the backend server. The backend server must validate this request by asking some user information in respect with the token to the Access Control tool. The backend server then receives some user information and can now match the received data with a user.

In the case where the authorization policy is not respected, the Access Control tool will inform the backend server that it has to reject the request made by the device, as depicted in Figure 26:

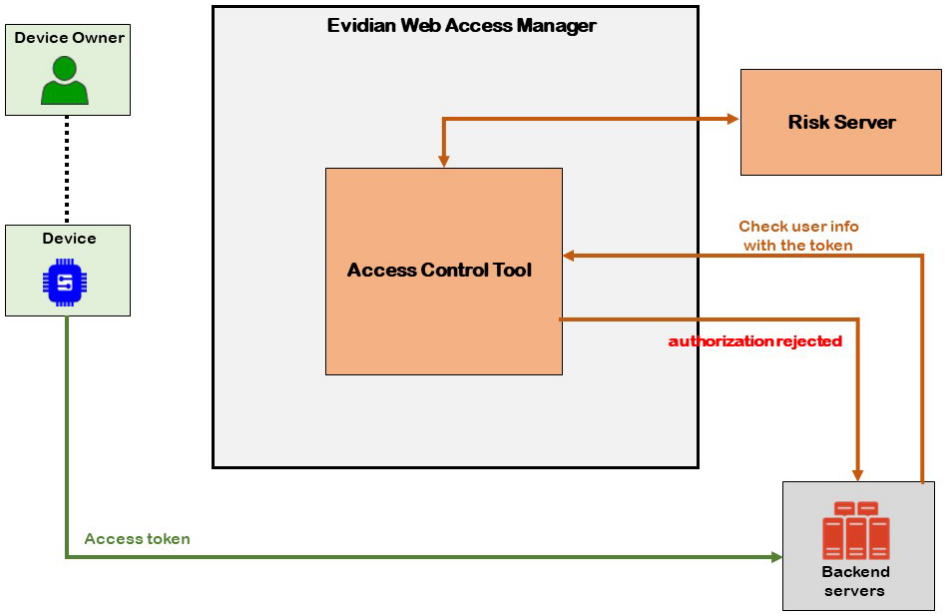

Figure 26. Context-aware Access control with WAM without reverse-proxy – Access denied

The sequence diagram for this working mode is described in Figure 27:

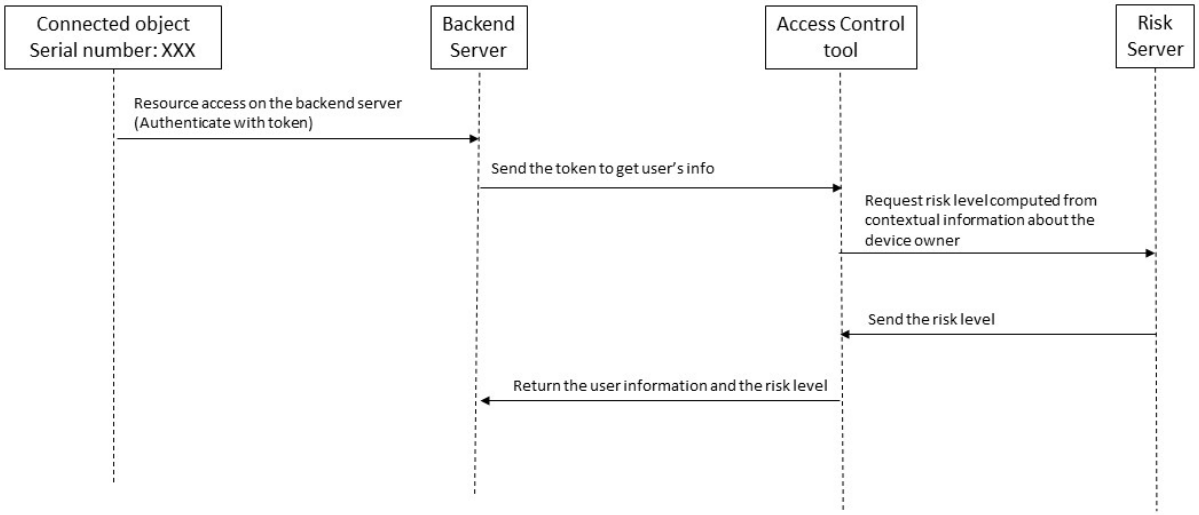

Figure 27. Context-aware Access control with WAM used as OpenID Connect IDP – Sequence diagram

The device uses its token to access the backend server (for example to push some data). The backend server checks the validity of the token and retrieves the device owner's consent scopes for this token by calling the userinfo endpoint of WAM (the userinfo endpoint from the Access Control tool API consumes a token to retrieve information on the user). WAM returns the user information (for instance: username, email, address) and the dynamic contextual risk level associated to the user. The backend applications can use this additional information to perform special actions.

4.2.4.4 Retrieve the user info and risk level from the device

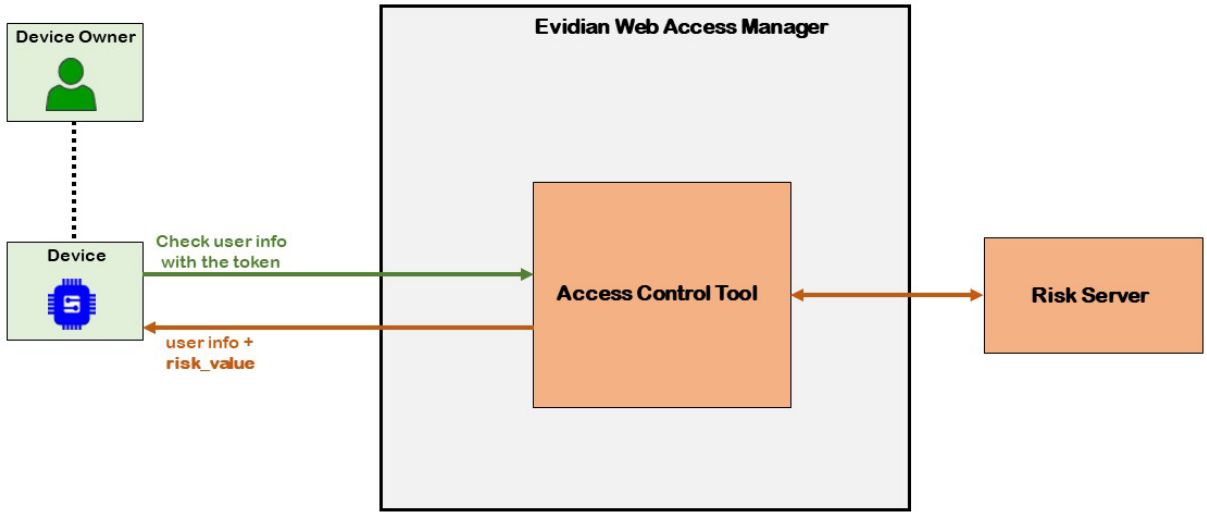

Figure 28. Retrieve the user info and risk level from the device

This mechanism depicted in Figure 28 enables the device to get some information on the user associated with it and its associated contextual risk level.

For example, the arm tensiometer needs to display on a little screen the name of the user and a dynamic data like the risk value. The device can periodically perform request to the Access Control tool to retrieve this data and display it on the screen.

The device performs a request to the userinfo endpoint of the Access Control tool with its access token. Then, the Access Control tool responds with the user information and the contextual risk level of the user associated with the device.

The response is a JSON containing the *User Information*, where the "security" field contains the risk level associated with the user.

Example:


```
{
"sub":"smith@Built-in User\u0027s Directory",
"name":"John Smith",
"preferred_username":"smith",
"given_name":"John",
"family_name":"Smith",
"middle_name":"",
"nickname":"",
"profile":"",
"picture":"",
"website":"",
"gender":"male",
"zoneinfo":"",
"locale":"",
"updated_at":"",
"birthdate":"",
```

```
"email":"",
"email_verified":false,
"security":{"risk":["10"]}
}
```

4.2.5 Tutorial

An instance of the Context-aware Access Control tool is available in SaaS mode at:

https://enact-caac-auth.evidian.com

It is accessible to users authenticated through a demo Evidian WAM server available in SaaS mode at:

https://enact-caac-my.evidian.com

A tutorial platform allows to create the following simulated components:

- Connected devices that need to access a back-end server; these devices are enrolled and associated with a user;
- A back-end server;
- Alarm buttons that provide a context information; these buttons are enrolled and associated with a user.

These simulated components are accessible through the demo SaaS WAM portal shown in Figure 29: https://enact-caac-my.evidian.com

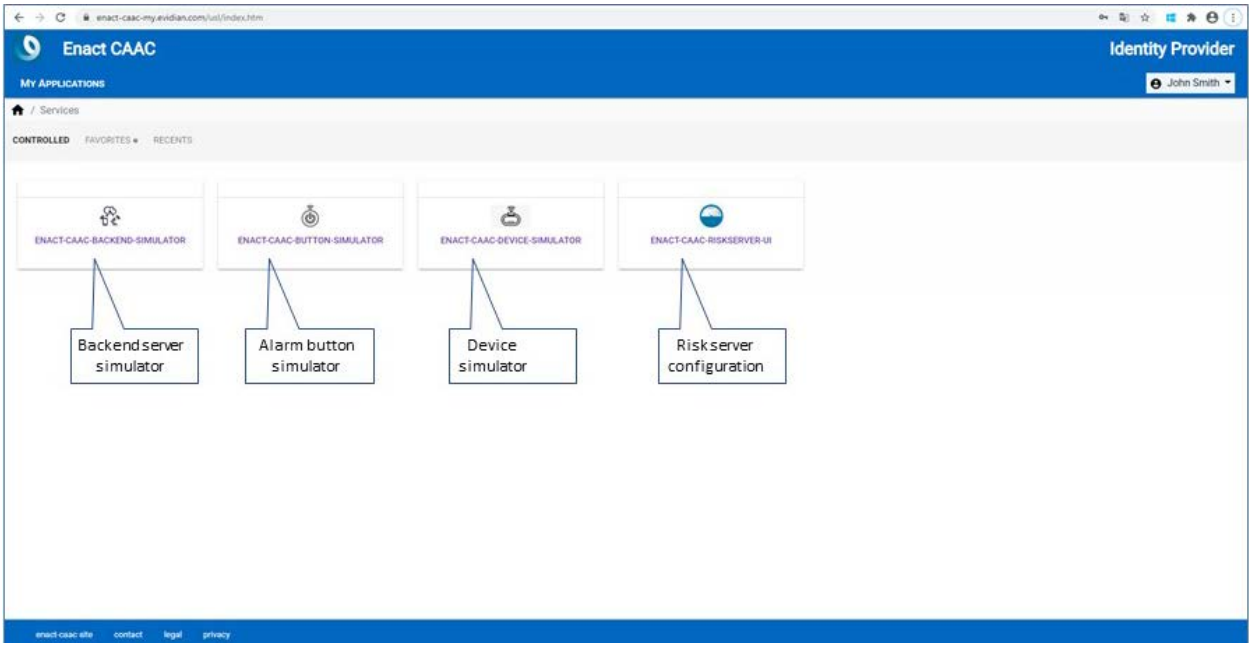

Figure 29. CAAC demo platform portal

This demo and test platform[15] contains also a simple risk server, intended to validate the architecture and to provide risk levels depending on both the contextual events sent by simulated alarm buttons and the audit events generated by WAM following the users' connections. A mock-up application (Riskserver-ui) is provided to be able to tune the taking into account of these events by the risk server. The provided features are:

[15] EVIDIAN Proprietary software. The user's guide is available on demand.

- Adding, deleting and modifying Risk levels.
- Management of weights linked to audit events.
- Management of weights linked to IP addresses.
- Complete management of the users and their associated buttons.
- Visualization of the history of the user's actions.

In addition, the demo and test platform provides features for scalability tests through the Alarm button simulator, in order to evaluate how much data the infrastructure can handle simultaneously.

4.2.6 Main innovation

The main innovation brought by the features offered by the Context-Aware Access Control Enabler can be summarised as follows:

- The solution provides one unique tool to control in the same way the access of all the IoT actors (end-users, services, devices, administrators) to the operated data and resources, for both IT and OT (operational technologies) domains.
- The solution adds dynamicity to the authorization decisions OAuth 2.0 produces, by injecting dynamic scopes in the standard Device flow.
- This allows to exploit contextual risk levels as dynamic attributes in the authorization mechanisms.
- Accordingly, the provided authorizations can be adapted based on a risk level computed from contextual data on the user and his devices, which allows context-aware dynamic access control behaviors.
- These mechanisms are available either through a reverse-proxy API gateway, or through an OpenID Connect Identity Provider.

The research approach behind this tool has been partially published in the following book chapter.

- Gallon, A., Rios, E., Iturbe, E., Song, H. and Ferry, N. Making the Internet of Things More Reliable Thanks to Dynamic Access Control. Book Chapter in Security and Privacy in the Internet of Things: Challenges and Solutions, IOS Press, 2020. ISBN: 978-1-64368-052-1, DOI: https://doi.org/10.3233/AISE200005

4.3 Diversity-oriented fleet deployment and operation of IoT systems

This section presents the new development of the DivEnact tool described in D4.2, Section 4.2.3 – code diversity. The new development goes beyond the original objective to only support software diversity among IoT devices, as a way towards IoT resilience.

Based on the analysis of the ENACT Use Cases, as well as literature reviews and case studies outside of the project, we recognize that the automatic deployment of multiple software variants into a fleet of IoT/Edge devices is a fundamental challenge in the DevOps for the typical IoT/Edge systems that consists of large number of "Edge" devices to which we need to deploy and refresh software on them. Keep the software diversity among these devices for the purpose of security and resilience is just one objective of such fleet-level automatic deployment.

There are also other more generic objectives, such as supporting the heterogenous devices with proper software variants on each of them. Based on this observation, we extend the DivEnact tool to a new concept – Fleet Deployment, which means that developers do not deploy a software update to individual devices, but instead, deploy the entire fleet, while the fleet deployment engine assign the multiple software variants (including the new updated version, but also its various configuration and the previous

versions that are still active) to all the devices in the fleet, and then launch the actual software installation or update individually and automatically on the devices.

This new section describes the concept of Fleet deployment and the DivEnact tool that implement this concept. The initial design of the DivEnact tool, as described in Section 4.2.3, is still the backbone of the current tool and all the features related to runtime code diversification as described in the previous section is still included in the current version of the tool. We will not repeat these existing contents, but focus on the new components of the tool that we developed since the previous deliverable, in the direction of the fleet deployment concept. The new components are:

- A GUI under the ENACT standard style for the complete fleet deployment function (including the diversity control features described before, through command-line UI), which is served as the dashboard for IoT DevOps teams and can be used together with GeneSIS as a deployment bundle for both IoT fleet and local subsystems.
- A novel fleet assignment approach based on model-driven engineering and constraint solver.
- An alternative fleet assignment approach based on the resource assignment problem and solver.
- An example based on the TellU Use Case.

4.3.1 Purpose and concept

With the rapid growth of the Internet of Things (IoT), extreme amounts of data are collected from the physical world. Due to network limitation and strict latency requirements, more and more data are processed close to where the data are generated. This led to the emergence of Edge computing, where software components are deployed on devices at the Edge of the network, such as gateways, routers, small base stations, etc. An Edge computing application normally comprises tens to thousands of distributed Edge devices, collectively referred to as a device fleet.

Edge application providers are essentially software developers and typically follow the modern DevOps practice to continuously add new features to their software running both on the Cloud and at the Edge. In this context, a practical challenge is how to automatically deploy the updated software to the many distributed Edge devices after each DevOps iteration.

In Cloud computing, most of the automatic deployment are solved by Infrastructure as Code (IaC): After each iteration, developers update the deployment script and call the IaC engine to deploy all the software as whole into the central Cloud. Unlike the centralized Cloud model, where computing resources are homogeneous, an Edge fleet consists of distributed and heterogeneous devices, which have different contexts in terms of hardware capacity, network connection, user preferences, etc. Developers need to maintain multiple variants of the software to fit such different device contexts. This raises the main problem for the automatic deployment of Edge computing applications, i.e., how to assign multiple deployments (each "deployment" is a unique composition of variant software components) to all the devices in the fleet, so that each device is assigned with a proper deployment that matches its context and, at the same time, the whole system meets its global goals, e.g., keeping the software diversity within the fleet and selecting a designated number of devices to trial a preview version.

To the best of our knowledge, there is no effective solution to this fleet deployment problem. The state-of-the-art Infrastructure as Code (IaC) tools automate the deployment of one application on one device, or a predefined set of devices. The mainstream IoT/Edge fleet management platforms offer to maintain multiple deployments, the fleet of devices and their contexts, but developers still need to manually designate which deployment goes to which devices. As a result, Edge application providers are left to manually assign the various versions of their software to hundreds or thousands of devices, which becomes a bottleneck for their DevOps process.

Aiming at this challenge, we propose the new concept of *fleet deployment*.

Fleet deployment is an automatic software deployment support for IoT/Edge applications, which allows developers to deploy software artefacts into the fleet of devices as an abstract whole, without concerning about the concrete devices and their contexts in the fleet. The automatic fleet deployment tool will maintain the devices and their contexts in the fleet, the software variants, and assign the variants to the appropriate devices depending on their contexts.

Fleet deployment is the cornerstone for DevOps of IoT/Edge applications that comprise a large number of similar but independent devices, with the TellU system as a typical example, which has hundreds to thousands of gateways deployed at the locations of their patients. Every time a new version or variant of the software is released, or a new device is included to the system, there is a need for a round of fleet deployment, with part or all the devices refreshed.

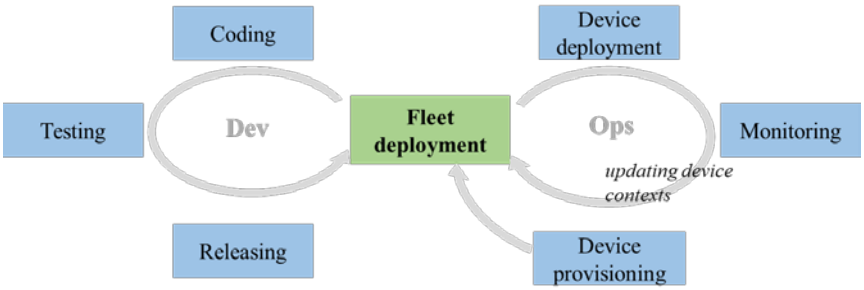

Figure 30. Fleet deployment in DevOps.

We implement the concept into a fleet deployment tool, which integrates the scattered proof-of-concept experiments we conducted in the first period (as reported in Section 4.2.3) and inherits the named DivEnact. As technical cores of the DivEnat tool, we conducted industrial research that applies model-based techniques to achieve automatic fleet deployment, by assigning multiple deployments to many devices in the fleet, without human interaction. The tool development is driven by the TellU eHealth Use Case and a set of experimental scenarios using the tool through a series of DevOps iterations shows that the approach is able to produce valid assignments, exempting developers from heavy manual effort, and to provide valuable feedback for planning the subsequent development.

4.3.2 The DivEnact tool and GUI

The DivEnact tool is designed and implemented following the established Model-View-Controller (MVC) design pattern for client-server application systems. The overall architecture is depicted in Figure 31, while the brief overview of the three main elements, as well as interactions with external components, follow below.

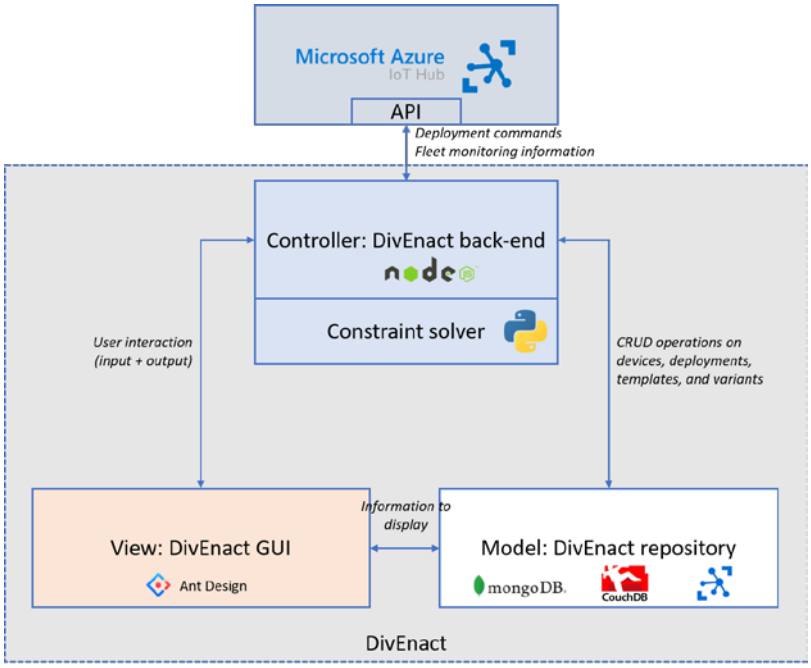

Figure 31. DivEnact architecture.

4.3.2.1 Model: DivEnact repository and Azure IoT Hub

The DivEnact knowledge base stores various deployment- and fleet-related artefacts and is modified by the DivEnact back-end model upon the user input received through the graphical use interface. The modification actions (CRUD – create, read, update, delete) are implemented on top of standard APIs and libraries. The model itself is spread across the following three repositories:

- **MongoDB** [51]: this database is installed locally, along with a DivEnact instance, and serves to store information about templates and variants unique to each application system.

- **CouchDB**[52]: this is a centralised repository for storing various ENACT artefacts, including deployment models used by DivEnact. In particular, the CouchDB database stores previously designed GeneSIS deployment models that are to be enacted on low-level IoT devices as part of the "last mile deployment".

- **Azure IoT Hub:** the cloud portal keeps track of registered devices in the fleet and existing deployments. The information obtained from the hub reflects the current state of all the devices through continuously updated digital twins, as well as deployments applied to these devices (i.e., software modules currently deployed and running on each device).

4.3.2.2 View: DivEnact graphical user interface

The DivEnact graphical user interface remains the main point of interaction with the user. The main functionality is structured across several tabs in the top menu, as depicted by Figure 32 and explained below.

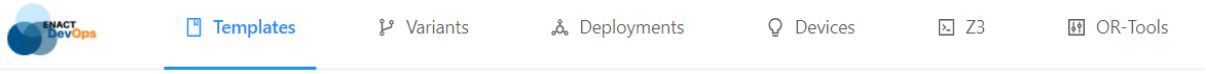

Figure 32. Functional tabs in the DivEnact user interface.

- **Templates:** lists the available deployment templates, stored on the local MongoDB. The tab also provides basic editing functionality to modify templates.

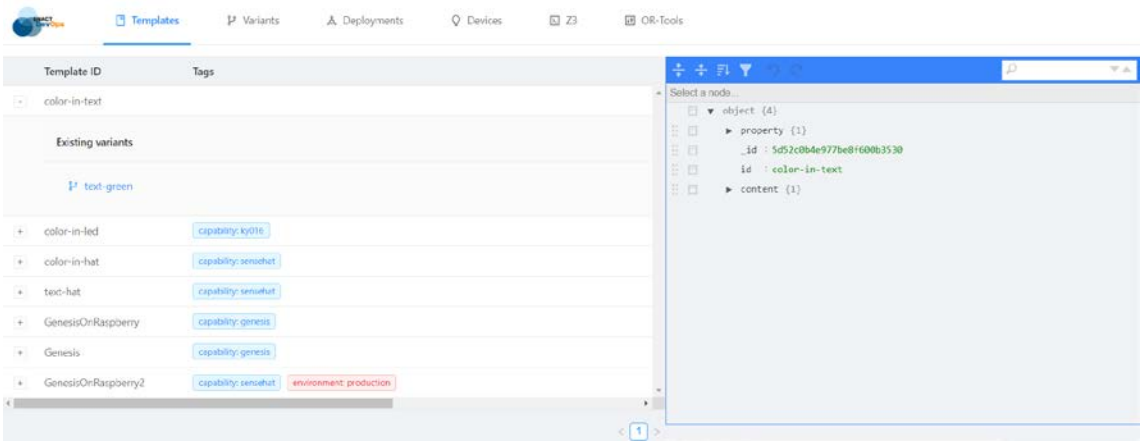

Figure 33 Template view

- **Variants:** lists the available variants (linked to templates) that can be parameterized and pushed for deployment to Azure IoT Hub. The tab also provides basic editing functionality to modify variants, including specifying a GeneSIS model for "last-mile deployment" available in the CouchDB central repository.

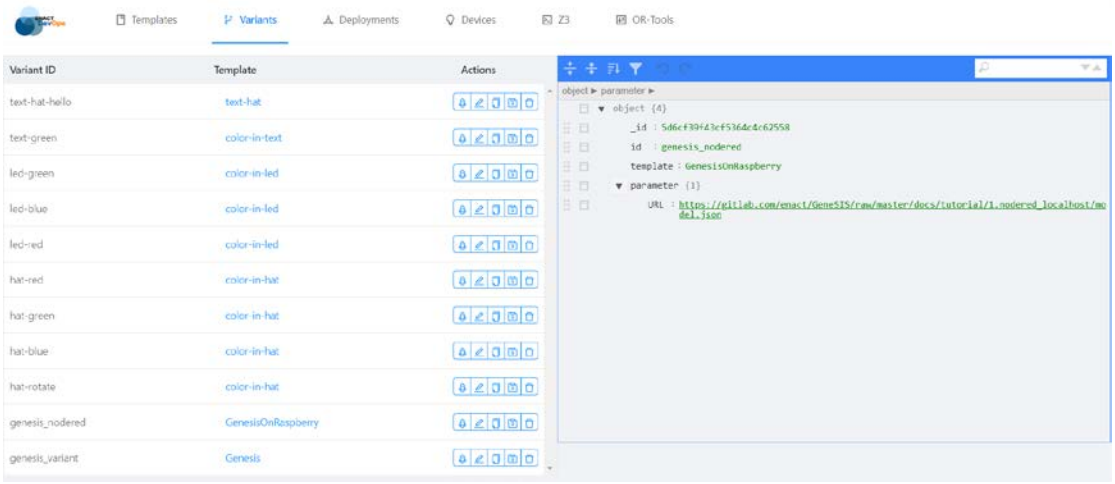

Figure 34. Variant view

- **Deployments:** lists the deployments, as fetched from Azure IoT Hub.

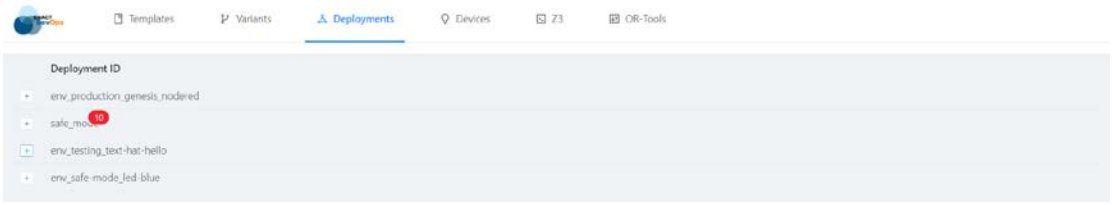

Figure 35. Deployment view

- **Devices:** lists the edge devices, as fetched from Azure IoT Hub. The tab also provides basic editing functionality to tag devices with certain properties.

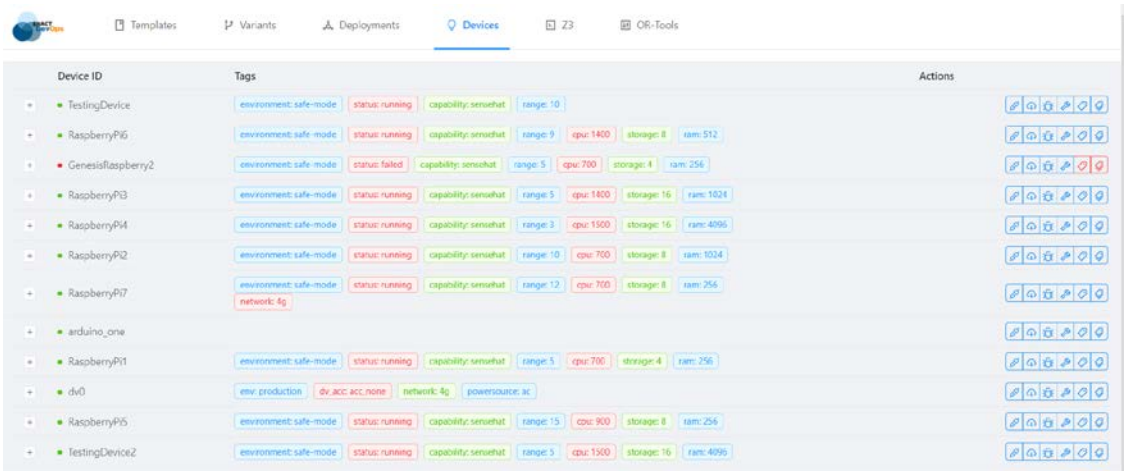

Figure 36. Device view

- **Z3:** provides functionality for designing assignment based on template/variant parameters and device properties. The designed assignment logic is fed to Z3 solver for solving and, upon approval, is finally pushed to Azure IoT Hub for deployment. This functionality is explained in more details in subsection 4.3.4.

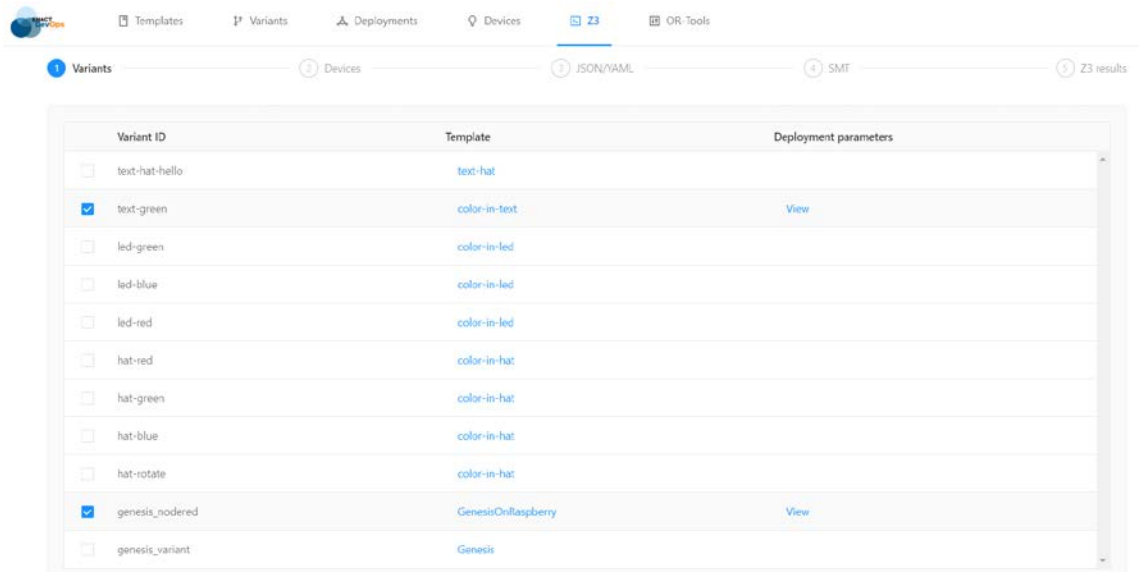

Figure 37. Constraint solving view

- **OR-Tools:** provides functionality for designing assignment based on template/variant parameters and device properties. The designed assignment logic is fed to OR-Tools solver for solving and, upon approval, is finally pushed to Azure IoT Hub for deployment (NB: for testing purposes only). This functionality is explained in more details in subsection 4.3.5.

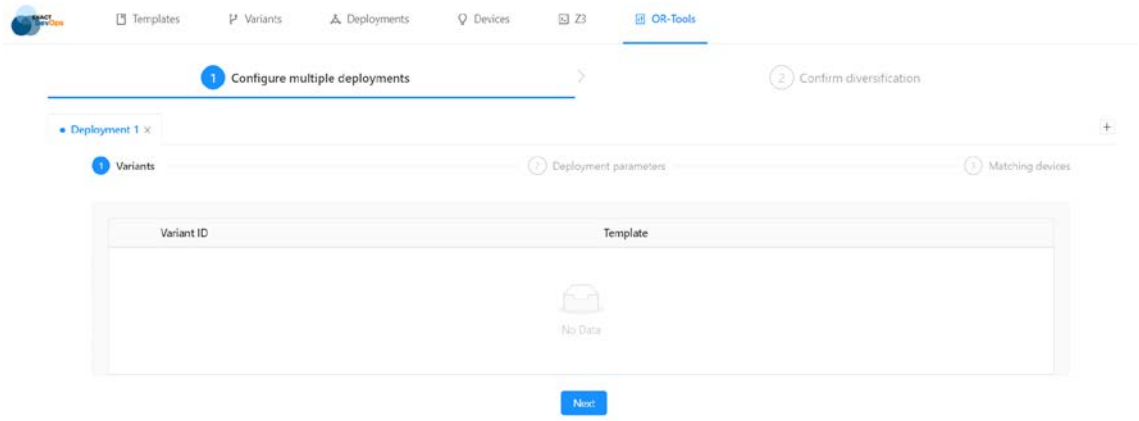

Figure 38. OR solving view

The graphical user interface of the DivEnact tool follows ENACT's common design template and is implemented using the Ant.Design [53] – a React UI library for enterprise-level products. It offers a wide collection of graphical elements, many of which have been used to create the DivEnact GUI. Many of the graphical elements implement an action – essentially, a RESTful invocation of the DivEnact back end.

4.3.2.3 Controller: DivEnact back-end

The back end of the DivEnact tool is implemented in Node.js. It receives RESTful requests originating from the user's graphical interface and manipulates the data model accordingly. It also interacts with the Azure IoT Hub API to update some information about the devices in the fleet and trigger deployments.

The back end also implements the actual diversification functionality (described in the next subsections) by receiving the input model from the user and passing it to the underlying Python script. Upon execution, the calculated solution is passed back to the user for the final approval.

4.3.3 The ENACT deployment bundle for IoT/Edge/Cloud continuum

DivEnact is implemented as a deployment solution for "edge computing", deploying software on a fleet of edge devices, typically gateways, which has relatively strong computation capacity and independent internet connection (which often means devices with a full-functional operating system, such as Raspberry Pi with a Linux distribution). A challenge remained is the "last-mile" deployment, i.e., deploying software into sensor and actuators which are so resource constrained that they do not have operating systems or container engines. DivEnact can be bundled with other IoT deployment tools to provide the capability of such last-mile deployment, and we implement such a bundle with GeneSIS, another tool from ENACT, described in D2.3.

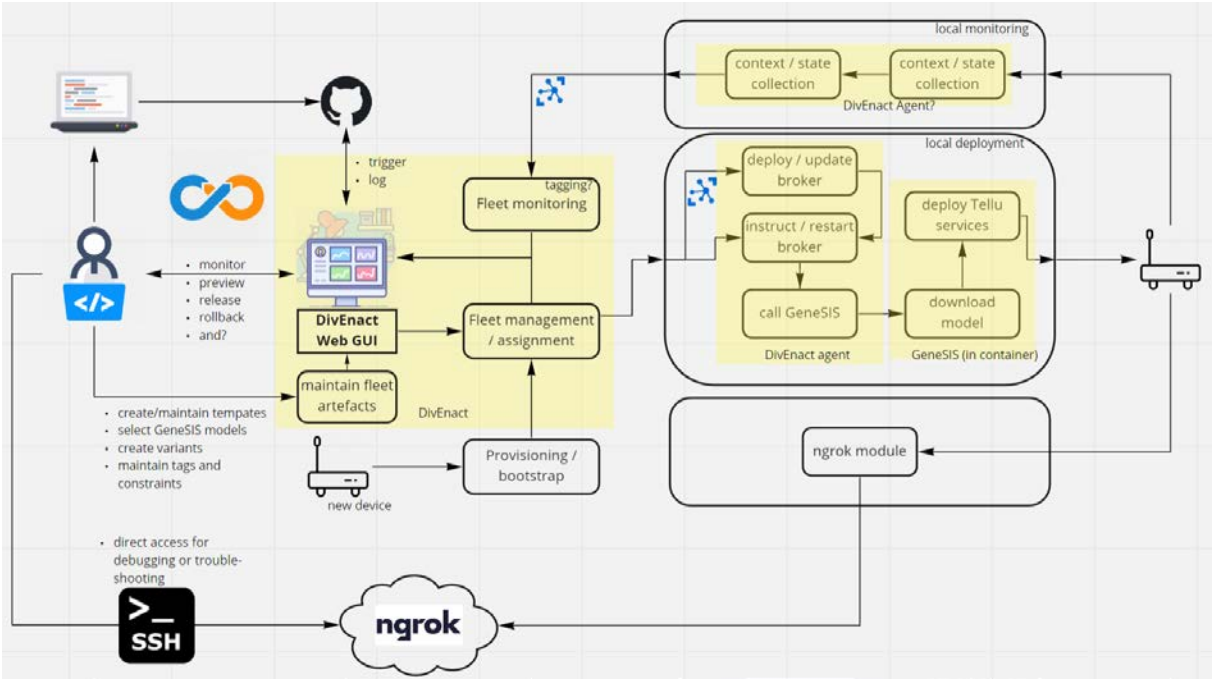

Figure 39. Activities supported by ENACT deployment bundle.

The figure above describes the typical activities supported by this "ENACT deployment bundle" with DivEnact, GeneSIS and some auxiliary 3rd party tools. The yellow shadows indicate the ENACT tools behind the activities.

Starting from the left-hand side. In a typical DevOps loop, developers change code to the IoT/Edge software, normally using an IDE. The change leads to a new version of the software, which they can choose to deploy to the device fleet. They can do this either manually through the DivEnact Web GUI, or automatically through a Continuous Delivery pipeline, such as GitHub Action. The operations in the fleet level are:

- Monitoring the list of devices and their contexts
- Add the new version of software as preview
- Release the review version into production
- Rollback specific devices into a previous version

Receiving these commands, the DivEnact GUI will invoke the fleet management and assignment modules in the backend to re-allocate the software among the fleet of gateways. For those gateways that are assigned with a different software version, the backend service will send the new deployment model to the DivEnact agent running on the gateway. Since the gateways are loosely coupled to the service, this deployment command from the backend to the gateway is communicated through the Azure IoT Edge. The DivEnact agent on the relevant gateway will further invoke the GeneSIS engine running on the same gateway to actually deploy the software, both on the gateway and on the devices associated to the gateway.

All gateways are registered to the Azure IoT hub service and the latter will monitor the status and manage the communication with them. In this project we focus on the fleet deployment and leave monitoring as a further work in the remaining period of the project.

Last but not the least, through DivEnact we can deploy a predefined module into a device, which set up a remote SSH service into the device through the *ngrok* cloud service[55]. In this way we can support the trouble shooting of the device when they are failed.

4.3.4 Fleet assignment using constraint solving

This section presents a research approach to address the automatic assignment of multiple software variants (deployment plans) to a fleet of devices, according to the contexts of the devices. We use a motivating example extracted from the TellU use case to explain the fleet deployment problem and after that explain our model-based approach to address this problem, with the help of the constrain solving technique. The main content of this section is published as a conference paper in the 20th International Conference on Model Driven Engineering Languages and Systems (MODELS) and awarded the best paper in the Practice and Innovation Track [28].

Motivating example

TellU provides Remote Patient Monitoring (RPM) services. For each of their customers (typically elderly people living at their own residences), they provide a healthcare gateway – a small single-board computer similar to Raspberry Pi – together with a set of medical sensors, cameras and wearable emergency beepers.

Each gateway collects measurements such as blood pressure, glucose and oxygen levels, etc., via Bluetooth, processes and aggregates the data, and sends them to the back-end cloud services. The patients and their nurses have access to the data via a Web interface and a mobile App. For some patients, TellU sends their technicians to mount the gateway on the wall, while other patients can opt for having the gateway delivered from the manufacturer and installing it themselves. Some patients choose battery-powered portable gateways to have a possibility to carry them with the essential sensors whenever they are outside their houses, e.g., for walking, taking medical examinations, or travelling.

A recent feature under development by TellU is *fall risk detection* based on machine learning, which combines live gesture detection via cameras together with other real-time physiological and environmental data to continuously assess the risk of patient falling down in a short term.

Since all the devices can be purchased off the shell, TellU is essentially a software vendor. Their main effort and focus is on developing the front-end and back-end software components, running on the gateway and the cloud, respectively. The development team continuously adds new features to the software and patches issues, following a DevOps practice.

During the process, they produce and simultaneously maintain multiple variants of the front-end software, which conceptually may be separated into vertical and horizontal ones. *Vertically*, they have development versions with the most recent features running on staging devices and released versions with mature features running on production-ready devices, already used by patients. In between of them, they regularly deploy preview versions with new features to a small set of selected patients, in order to collect feedback from real users. *Horizontally*, they maintain software variants that fit gateways with different setups and contexts. For example, if a variant has the machine learning (ML) module for fall detection running in the back-end, then it only fits those gateways that are connected to WiFi network, not 4G, since they need to send images to the cloud. Alternatively, the variant with the ML module running fully on the edge only fits the gateways that are mounted on the wall, since the heavy computation load implies risks of overheating the self-installed gateways (which usually do not have proper cooling/ventilation facilities). As an option, some patients may choose to buy an additional hardware accelerator (typically in the form of a pluggable USB dongle, such as Google Coral [56]) which can reduce the computation load on the gateway, but it requires an application variant with a re-compiled ML module).

Since no variant fits every device, TellU has to maintain multiple variants (Since each software variant represents a unique way of deploying the software, throughout this section, we use software variants and deployments interchangeably), and assign them to the devices according to their contexts.

In summary, TellU maintains a fleet of about 500 gateways for their patients and additional 10 local installations on their own premises. The gateways have different contexts, in terms of network, mounting, accelerator, etc. The development team maintains around 3-10 different variants of the front-end software. The TellU team expect to conduct DevOps cycles on a daily basis, retiring parts or all of the active variants and introducing new ones. However, since there is a mass re-deployment after each cycle involving many or sometimes all the devices, and it is not fully automatic at the moment, they cannot achieve such high frequent DevOps cycles. Automatic fleet deployment becomes a bottle net for the development team.

Fleet deployment: concepts and challenges

Fleet deployment is a problem of automatically deploying software on many functionally similar devices, while the actual software deployed may vary from device to device, based on the context. The figure below illustrates the conceptual architecture of a fleet deployment framework, with three main components:

- Fleet monitoring, which manages the lifecycle of all the devices and collects the context for each device;
- Fleet assignment, which maintains active candidate deployments and assigns a deployment plan to each device.
- Device deployment, which enacts the assigned deployment to each device, by installing software modules according to the deployment plan.

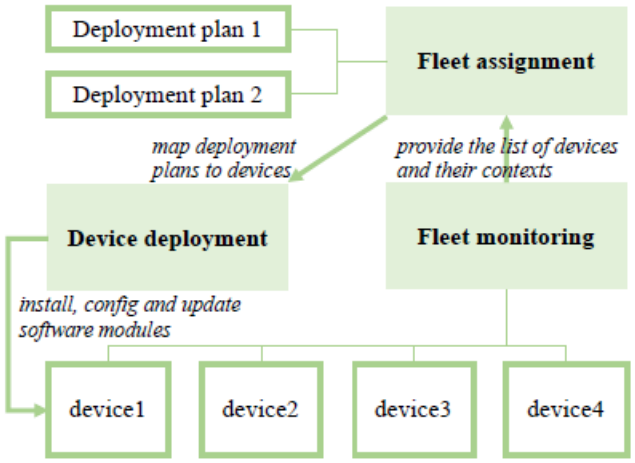

Figure 40. Architecture of a fleet deployment framework.

There are several commercial IoT/Edge development platforms providing fleet monitoring support, such as Azure IoT Edge. At the same time, device deployment is essentially an Infrastructure as Code (IaC) problem—i.e., developers provide a model specifying where to obtain the software modules their libraries, and the corresponding engine will deploy them accordingly on the Edge device. Mainstream IaC solutions, such as Ansible, also supports remote deployment. However, fleet assignment is still an open question.

The research in ENACT focuses on fleet assignment, which can be abstracted as a problem of finding the function:

$$d: Dv \rightarrow Dp \cup \{\perp\}$$

where Dv is the set of all devices and Dp is the set of active deployments. If no proper deployment can be found for a certain device dv, then $d(dv) = \perp$. This function d is the main output of fleet assignment, indicating what software should be deployed to each device. In addition, the assignment may also involve a set of functions for the configuration of deployments on each device:

$$c_x: Dv \rightarrow V_x$$

It passes additional parameters to each assigned deployment.

For example, $c_{\{ie\}(dv_1)} = \top$ means the device dv1 is configured in the way that machine learning is running on the gateway (where "ie" means *Intelligence on Edge*).

Fleet assignment is challenging, because it must satisfy the following requirements for all devices, at the same time:

R1. The software modules that are assigned to a device must match the particular device capabilities, including CPU architecture, memory volume, hardware accelerator, etc.

R2. A deployment that is still under development can be only deployed to staging devices. At the same time, developers may want to deploy the preview version on a designated number of production-ready devices for direct feedback.

R3. The deployment assigned to a device should have reasonable resource consumption, e.g., low CPU usage for a device powered by battery. Note that the same deployment may have different consumption on different devices, influenced by offloading, hardware acceleration, etc.

R4. A typical objective is software diversity, i.e., even distribution of deployment plans among the devices for the sake of resilience, robustness and security.

In practice, an Edge computing application may involve many devices that may run with various contexts, and thus can be deployed in multiple alternative ways. In such case, fleet assignment must handle multi-dimensional constraints for a large collection of objects.

Research approaches to the automatic searching of solutions under complex constraints have resulted in the emergence of multiple theories and tools, collectively referred to as Boolean Satisfiability Problem (SAT) [29], and more recently Satisfiability Modulo Theories (SMT) [30]. Given the various constraints and conditions, employing constraint solving theories and tools is a promising approach towards a fully automated solution for fleet management. However, a main barrier is the gap between the abstract SAT or SMT models and the concrete representation of devices and deployments obtained from the existing platforms for fleet monitoring and device software deployment.

Model-based fleet deployment

We employ a model-based approach to automate the fleet deployment problem, using meta-modelling and model transformation to bridge the gap between the abstract constraint solving theory and the concrete Edge computing platforms. The figure below illustrates the overall architecture of the approach.

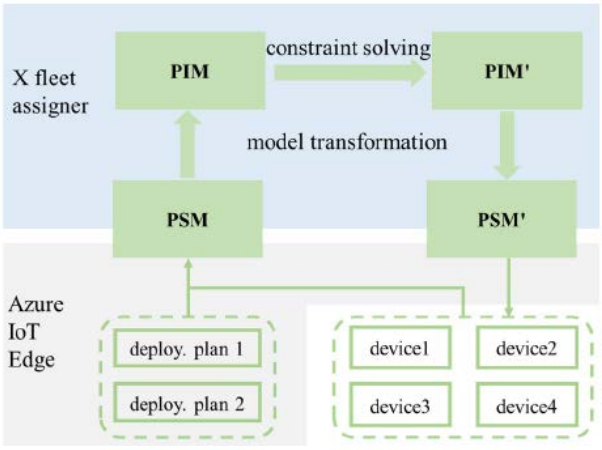

Figure 41. Model-based fleet deployment.

We obtain from the platform a PSM (Platform-Specific Model) that represents the available devices and deployments. We transform a PSM into a PIM (Platform-Independent Model), which represents the essential information, such as devices, deployments and their attributes, using a simple and standard notation, agnostic with regard to the platform. This original PIM, essentially representing the information from fleet monitoring, is not a complete model yet, since the mapping between devices and deployment plans is missing, together with some run-time configurations. We apply constraint solving on this PIM to automatically search for the missing part. The solution provided by the constraint solver is a new PIM' with the missing part. We then transform PIM' into a new PSM', which will be finally used by the platform to install the software modules. Transforming back to PSM is necessary since it is the format that the fleet management platform understands.

The approach itself is generic across multiple Edge computing platforms -- i.e., for different platforms, we need to define different PSMs and the model transformation, while the PIMs and the constraint solving approach remain the same. As reported below, in the context of this paper, we use Azure IoT Edge as the target platform to implement a Fleet Deployment tool and demonstrate the viability of the proposed approach.

The meta-models and the model transformation are straightforward, and we use the sample models below to explain them.

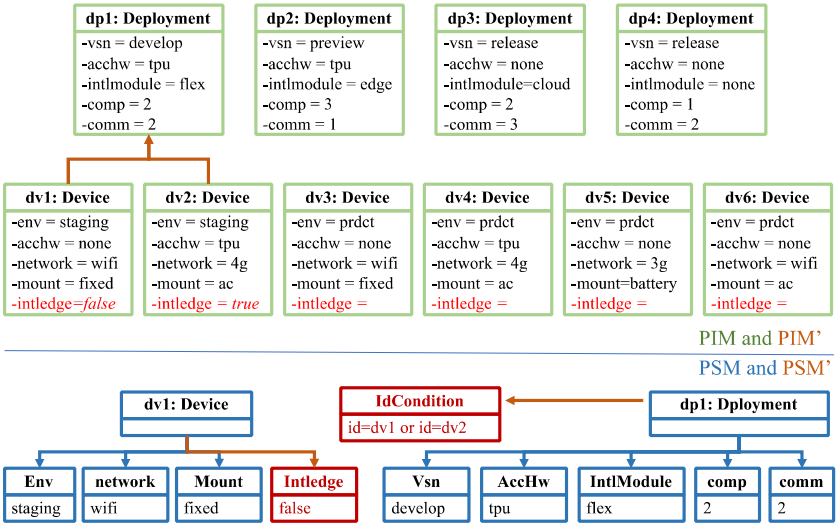

Figure 42. Sample models for fleet deployment.

According to the PIM, the system comprises of 6 devices. The second device (dv2) has a Tensor Processing Unit (TPU) as accelerator, connected by 4G and powered by A/C. Together with dv1, the two devices are used for staging. The other four devices are used in production.

We have 4 deployments under maintenance: dp1 is still under development and supports TPU and flexible offloading. The worst-case computation and communication levels are both medium (2 out of 3). Device Dp2 is a preview version, which hosts the machine learning module only on the Edge and supports TPU. The worst-case computation requirement is high, but communication requirement is low, since the analytics is done on the Edge. Attributes (intledge) and reference (deploy) highlighted in red do not exist in PIM, but will be added in PIM'. In the figure, we only show the partial assignment for the first two devices. We only show part of PSM, with one device and one deployment with attached attributes represented as tags. In PSM (the original elements in PSM plus the additional ones marked in red in the same figure), one new tag is created and the deploy reference is represented as IdCondition.

Fleet assignment using constraint solving

We employ SMT constraint solving and an existing implementation, Z3 solver [31], to achieve automatic fleet assignment. To do so, we represent the PIM as an SMT problem and define a set of hard and soft constraints according to the domain knowledge. The solver will automatically search for the missing values in the PIM, according to the constraints.

An SMT problem is a decision problem for logical formulas on a combination of background theories. A basic built-in theory for SMT is the *uninterpreted function theory*. An uninterpreted function is a function that is defined with only domain and codomain, without an interpretation on how the domain is mapped to the codomain. The building blocks of an SMT problem are a series of such uninterpreted functions, on which we can define first-order logic formulas as constraints. We transform a PIM into an uninterpreted function theory problem based on the following rules, explained using the sample PIM.

- All devices form a set $\mathrm{Dv} = \{\mathrm{dv}_1, \ldots, \mathrm{dv}_6\}$ and all deployments form a set $\mathrm{Dp} = \{\mathrm{dp}_1, \ldots, \mathrm{dp}_4\}$.
- The deploy relation is represented by a function d: $Dv \rightarrow Dp \cup \{\perp\}$, as discussed before.
- Attributes in primitive types are represented as functions to the corresponding types, e.g., $cp: Dp \rightarrow N$ and $ie: Dv \rightarrow B$, for *comp* and *intledge*, respectively.
- The attributes whose values are within a particular set, are represented by functions with codomains as the set of all the valid choices. For example, the *version* attribute is a function $vsn: Dp \rightarrow \{develop, preview, release\}$.

A constraint solver can automatically search for an interpretation for each of the uninterpreted functions that we defined above. The interpretation of function *d* (i.e., giving a *dp* for every *dv*) and the interpretation of *ie* (assigning either true or false for every *dv*) are the result for fleet assignment. The searching process will be guided by a set of constraints from: (i) the current model status, i.e., the existing attribute values; (ii) the domain-specific knowledge; (iii) global optimization objectives. In the rest of this section, we will present the three types of constraints, respectively.

In the current PIM, the devices and deployments already have existing attribute values, which serve as the basis for fleet assignment. Therefore, we translate all the current attribute values into hard constraints. The example below shows how we generate the constraints from the current state of *dp1*:

$$vsn(dp1) = develop \wedge\ pac(dp1) = tpu \wedge ie(dp1) = flex \wedge\ cp(dp1) = 2 \ldots.$$

We generate the constraints automatically by iterating over all the devices and deployments in the PIM, reading the attribute values and generating the equations accordingly.

We define a set of hard constraints based on the domain knowledge about software deployment on Edge devices. These constraints are related to the requirements R1 to R3. To simplify the constraint definition, we first introduce 3 auxiliary functions.

$$acc: Dv \rightarrow B,\ \ vcp: Dv \rightarrow N,\ \ vcm: Dv \rightarrow N$$

Here *acc* indicates whether the hardware accelerator will be used on a device, and *vcp* and *vcm* are the actual levels of computation and communication resource consumption on the device, which is determined not only by the levels defined in the deployment, but also by the configuration of the device, such as whether the machine learning module is executed on the Edge device or on the cloud (*ie*) and whether the accelerator is used (*acc*).

We start from the constraint related to the development stage (R2): any deployment with software under development should be only deployed to a staging device:

$$\forall\, dv \in Dv, dp \in Dp: d(dv) = dp \wedge vsn(dp) = develop \rightarrow\ env(dv) = staging$$

The next set of constraints requires that the resource consumption of a deployment should match the capacities of its target device (R1 and R3).

The first step is to define how the actual consumption of each device is determined by the theoretically worst-case consumption defined by the developers.

$$\forall\, dv \in Dv, dp \in Dp: d(dv) = dp \rightarrow$$
$$vcm(dv) =$$
$$\quad cm(dp) - 1 : im(dp) = flex \wedge ie(dv)$$
$$\quad cm(dp): otherwise$$
$$vcp(dv) =$$
$$\quad cp(dp) - 1 : im(dp) = flex \wedge \neg\, ie(dv)$$
$$\quad cp(dp) - 1 : acc(dv)$$
$$\quad cp(dp) : otherwise$$

For communication, if a deployment allows flexible offloading of the ML model, then running the model on the Edge device would reduce the requirement on communication, since the raw data will be consumed locally without being transferred to the cloud. For computation, there are two cases that the actual computation requirement on the gateway will be reduced: (i) When offloading is supported and the model is not executed on the Edge, the computation requirement will be reduced, since the computation task is offloaded to the cloud; (ii) When hardware accelerator is used, the computation requirement is reduced, since the task is offloaded to the accelerator.

The second step is to regulate when we can offload the ML module to the cloud or the hardware accelerator.

$$acc(dv) \Leftrightarrow ie(dv) \wedge vac(dv) = pac(dp) \neq none$$

$$ie(dv) \rightarrow im(dp) \in \{flex, edge\}$$

$$ie(dv) \Leftarrow im(dp) = edge$$

Offloading to accelerator happens (i.e., $acc(dv) = \top$), *if and only if* the ML module is allocated to the device (i.e., ie(dv)) and the type of accelerator supported by the deployment match the accelerator attached to the device. The equivalence statement means that whenever it is possible, we will opt for acceleration, which is the most reasonable for our use case. The ML module is executed at the Edge (i.e., $ie(dv) = \top$), only if in the deployment the ML module is defined as flexible or running on the Edge. On the other hand, if in the deployment the ML module is defined to be only running on the Edge, then on the device, ie(dv) must be on. It is worth noting that when $im(dp) = flex$, ie(dv) can be either true or false, which is a decision to be made by fleet assignment.

The last set of constraints describes how the device contexts determines what level of resource assumption is allowed.

$$\forall\, dv \in Dv:$$

$$nw(dv) = 4g \rightarrow vcm(dv) < 3$$

$$nw(dv) = 3g \rightarrow vcm(dv) < 2$$

$$mt(dv) = ac \rightarrow vcp(dv) < 3$$

$$mt(dv) = battery \rightarrow cvp(dv) = vcm(dv) = 1$$

The rationale behind these constraints in the sample domain is threefold. First, the worse network a device has, the lower communication level it can support. Second, if the device is not mounted professionally, it should not run software modules with the highest computation requirement. Finally, if a device is powered by a battery, it should opt for the lowest computation and communication consumption.

In addition to the hard constraints that must be satisfied by the assignment, we also define a set of soft constraints to guide the assignment towards optimized global distribution (Requirements R2 and R4).

The solver can choose to violate a soft constraint, with an agreed penalty, and it aims at the minimal total penalty for the solving result. In our case, we used the following three groups of soft constraints.

The first group of software constraints suggests each device to be assigned with a deployment. For each device, we generate a soft constraint for its deployment to be not empty, with a penalty of 50.

$$(d(dv_1) \neq \perp \ (p{:}\,50),\backslash;\ldots,\backslash;\ d(dv_6) \neq \perp)\ (p{:}\,50)$$

The second group consists of only one soft constraint, which instructs the solver to select 20\% of all production devices to trial the preview deployments (Requirement R2). The left side of the equation counts all the devices assigned with a preview deployment, while the right side counts all the devices in production and multiplies it by 0.2.

$$(\ \ |\{dv \in\ Dv \mid vsn(d(dv)) = prev\}| = \lceil\ 0.2 \cdot |\{dv \in\ Dv \mid env(dv)\ = prod\}|\rceil)\ \ (p{:}\,100)$$

The third group of soft constraints aims to evenly distribute the deployments in order to maintain the diversity of deployments in the whole group (R4). In the two constraints defined below, we count the number of devices that are assigned with each deployment and suggest the number to be close to the average number of devices per deployment (i.e., within the scope of $\pm$20\% of the average number).

$$|\{dv \in\ Dv\,|d(dv) =\ dp_1\}| > 0.8 \cdot \frac{|Dv|}{|Dp|}\ \ (penalty = 20)$$

$$|\{dv \in\ Dv\,|d(dv) =\ dp_1\}| < 1.2 \cdot \frac{|Dv|}{|Dp|}\ \ (penalty = 20\}$$

This is a relatively low-cost way to achieve even distribution, compared to other complex methods, such as minimizing the total variance (i.e., $\min \sum\ \ (\tau_i\ - avg(\tau)\ )^2\ /\ n$, where τ_i is number of devices for dp_i).

We use the Z3 SMT solver to automatically search for a solution for the uninterpreted functions under the constraints. We define the SMT problem in Z3Py, the Python interface of Z3, feed the defined constraints into the solver as input and extract the results. The code example below demonstrates a simplified example to illustrate the whole process. We first define an enumerated type Dp to represent the set of all deployments (Line 1), followed by an uninterpreted function d for deploy (Line 3). The definition for Dv is similar and is therefore omitted in this example. Function env links Device to another enumerated type representing the environment choices (Lines 4-5), and cp is a function from Deployment to integer. After defining the functions, we instantiate a solver (Line 7) and add constraints to it. We give two examples to show how to program the constraints that we presented earlier.

The first one is a hard constraint indicating that a "develop" deployment can be only deployed to a "staging" device (Lines 8-10). The second example is a group of soft constraints: For each deployment, we generate a constraint stating that the total number of devices assigned with it should not be more than 120 % of the average number (Lines 11-13). After feeding the solver with all constraints, we launch the solving procedure (Line 14) and then, as an example, check for a deployment assigned to Device No. 4.

```
Dp, dps = EnumSort('Dp', ['dp1', 'dp2', 'dp3', 'dp4', 'nodp'])
d = Function('deploy', Dv, Dp)
Env, [staging, prod] = EnumSort('Env', ['staging', 'prod'])
env = Function('env', Dv, Env)
cp = Function('comp', Dp, IntSort())
solver = Optimize()
solver.add(ForAll([dv, dp], Implies(
  And(d(dv) == dp, vsn(dp) == develop), env(dv) == staging)))
for i in dvs:
  solver.add_soft(Sum([If(d(j)==i, 1, 0) for j in dps]))
```

```
    < len(dv)*1.2/len(dp), 20)
solver.check()
assert dp2 == solver.model().eval(d(dv4))
```

Application scenarios

We evaluate the effectiveness of the approach by applying it to a DevOps scenario that simulates the real development activity at TellU and observing how it assigns the deployments at different stages during the development process. The sample scenario demonstrates that:

- Automatic fleet assignment is useful for developers to accelerate the DevOps process.
- The constraint solving approach can produce valid assignment for developers.

The scenario consists of a series of DevOps iterations, each of which yields a new version of the application, changes the status of existing versions, and/or retires some old versions. Through these iterations, the developers gradually create, test and release 6 versions of the RPM application, in order to introduce and improve the new fall detection feature with a machine learning module. Table below summarizes these versions and their resource consumption. Each version corresponds to one deployment.

Dep.	Summary of features
A	Base version, no ML (comm:1, comp: 1)
B	ML running on cloud (comm: 3, comp: 1)
C	ML running on gateways (comm: 1, comp: 3)
D	Flexible ML offloading (comm: 3, comp: 3)
E	Improve C by supporting accelerator (comm: 1, comp: 3)
F	Improve D, offloading + accelerator (comm: 3, comp: 3)
G	Improve E with lighter model (comm: 1, comp: 2)

Figure 43. Evaluation scenarios

The next table summarizes the automatic assignment result after each iteration. The first column lists the set of deployments maintained by the developers after each iteration. We use the vertical bar sign | to divide the deployments into three groups, according to the different development phases (or vsn as we named it in the model). For example, ``AB|C|D" means deployment ``A" and ``B" are released to production, ``C" is under preview and ``D" is under development. As iterations go on, more deployments are released and go into preview or deployment phases.

	deploy	staging devices				production devices																			
#	ments	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	A||	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
1	A||B	*B*	A	B	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	A|B|C	C	A	B	A	A	B	A	A	A	B	A	B	A	A	B	A	A	A	A	A	A	A	A	A	A
3	AB|C|D	D	A	B	A	A	B	A	C	C	B	A	B	C	A	B	A	C	A	A	A	A	A	A	C	A
4	ABC|D|E	C	E	B	A	D	D	A	D	D	B	A	B	D	C	B	A	C	A	A	A	A	A	A	C	A
5	ABCD|E|F	F	E	D	A	D	C	A	E	D	A	A	D	E	C	E	A	C	E	E	A	A	A	A	C	A
6	ABDEF||	D	E	B	A	D	B	A	D	F	B	A	F	F	E	B	A	E	E	F	A	A	A	A	D	A
7	BDEF||	B	F	E	-	D	E	-	D	D	F	-	F	D	E	D	-	E	E	F	-	-	-	-	F	-
8	DEF||	D	F	E	-	F	E	-	E	E	F	-	E	F	E	F	-	E	E	F	-	-	-	-	F	-
9	DEFG||	D	E	F	-	D	F	G	D	D	F	G	D	E	E	D	-	E	E	F	-	G	G	G	F	-

Figure 44. Assignment results for the evaluation scenarios

The rest of the table schematically illustrates how the proposed tool automatically assigns the deployments after each iteration to the 25 devices, including 4 staging devices and 21 production devices.

The devices have different contexts in terms of mounting, network and accelerator, based on sampling with the actual devices maintained by the company. We omit the actual profile of each device, since the important part is to evaluate how the approach provides valid assignments as a whole, rather than to check if each single assignment satisfies all the constraints (we do not need to validate the correctness of Z3 solver). We go through the assignment results during the entire scenario, step by step, to show how the automatic fleet deployment benefits developers. For each step, we give a brief description of the development purpose, followed by how the automatic assignment results support this purpose.

1. Baseline deployment. In the beginning, A is the only valid deployment and has the lowest resource requirements and therefore all devices are assigned with this deployment.

2. First attempt with machine learning (ML). Developers first implement B with the ML module running in the cloud, which requires devices with WiFi due to the high communication consumption. Two staging devices were selected after iteration #1 to test B internally, but afterwards only 3 were picked for preview in #2 (expected to be 20%, or 4 devices) and the same 3 after B is released (#3), since WiFi is not widely used by patients -- most of them would skip configuring WiFi and keep using the default 4G network. The developers get the feedback that running ML only in the cloud has limited usage for their users.

3. Migrating ML to the Edge. Deployment C with ML running on the Edge (tested, previewed and released through #2-#4) is complementary to B: It requires fixed mounted devices but not necessary WiFi. Comparing iteration 2 and 3, C is deployed to devices that B was not assigned to, which is in line with the developers' expectations.

4. Support Cloud-Edge offloading. The next version D allows flexible offloading. Due to the networking overheads, it has some extra cost and is therefore not a complete substitution to B and C. The developers choose to keep all the three deployments. They would expect an even distribution of A-D among the production devices at #4, but the assignment result has too many A's (10 devices in total), which indicates that there are still many devices that cannot run B, C or D.

5. Utilize hardware accelerators. Deployment E and F add accelerator support to C and D, respectively, by adding an ML module compiled for Edge TPU. E and F still keep the original ML module, so that they can also run on devices that do not have a TPU (in that case, the computation consumption would not be lowered). This means that E can completely replace C. Therefore, when releasing F (#6), the developers also retire C. The new assignment result does not increase the number of A's, which leads the developers to hypothesize that the remaining 10 devices with A could not run any of the current deployments with ML.

6. Try to retire the base version. To confirm the hypothesis, the developers attempts to retire A (#7) and, as expected, 8 production devices end up with no deployments, and so is one staging device (Device 4). They go on to retire B, based on an observation back in #5 that when the assignment tool tried to recruit a sufficient number of devices to preview E, it removed B completely, which means B is replaceable by the subsequent versions. The result at #8 also confirms this, as the number of un-deployed devices is the same. As trial steps in #7 and #8, the developers run assignment without executing the subsequent device deployment, to avoid production devices having no deployments.

7. Optimize ML module. By examining the profile of undeployed devices, the developers realize that the computation power is the key problem and therefore they simplified the ML module and produce G with lower computation requirement. By releasing G (#9, we skip the iterations for testing and previewing G, for the sake of space limitation), only four devices remained undeployed. This can be further solved by sending accelerators to these devices, so that eventually all the devices can be equipped with the new ML feature.

In summary, the scenario demonstrates the following benefits for developers.

- **Automation**. After each iteration, developers do not need to manually select matching devices for preview and release.
- **Handling complexity**. Maintaining multiple deployments in parallel often needs to re-assign existing deployments to make space for new ones. Taking the transition from #4 to #5 as a simple example, the assigner has to change Device 12 from B to D, in order to release Device 13 to preview E. When many devices are involved in such re-arrangement, it would be too complex for manual work.
- **Testing**. Running fleet assignment without the device deployment step is an efficient way to test the composition of deployments, as is shown by the attempt to retire A and B.
- **Feedback**. Knowing that the assignment tool will satisfy the constraints and will aim at a balanced distribution, developers can use the results to understand the missing parts of the current deployments and plan for subsequent development.

4.3.5 Fleet assignment using resource assignment algorithms

This section presents an alternative research approach to address the automatic assignment of multiple deployment plans to a fleet of devices, according to the contexts of the devices. The main content of this section is part of a paper accepted for publication in the 9th International Workshop on Cloud and Edge Computing and Applications Management (CloudAM).

Unlike the traditional cloud model, where computing resources are relatively homogeneous, an edge fleet consists of distributed and heterogeneous devices, which have different contexts in terms of hardware capacity, physical environment, network connection, user preferences, etc. To address such heterogeneous contexts, software components are also becoming increasingly diverse, and edge software providers often simultaneously maintain multiple active versions of the same application. The existing tag-based fleet assignment mechanisms offered by IoT cloud platforms are sufficient for rather simple and small-scale scenarios with a few characteristics, which are expressed as literal values. However, they are not expressive enough to address software assignment scenarios with hundreds and thousands of devices, each having its own continuously changing context consisting of various numeric properties – on the one hand, and multiple software components with several active versions – on the other. In these circumstances, correctly assigning software components in a scalable and timely manner becomes an important challenge. Thus, using the default available tag-based assignment mechanisms, albeit possible, would result in a complex collection of chained fine-grained equality conditions – an error-prone, cumbersome and difficult-to-maintain solution.

4.3.5.1 Fleet assignment as a linear integer problem

In more formal terms, fleet assignment can be formulated as a linear integer program. We assume having $\boldsymbol{m}$ devices, $\boldsymbol{dv_1}$ through $\boldsymbol{dv_m}$, and $\boldsymbol{n}$ deployment plans, $\boldsymbol{dp_1}$ through $\boldsymbol{dp_n}$. Each device-deployment plan pair is associated with a local suitability $\boldsymbol{ls_{i,j}}$, which is a numeric quantifier indicating how well a device $\boldsymbol{dp_i}$ is suited to be assigned with a deployment plan $\boldsymbol{dp_j}$. Accordingly, the goal of the assignment is to select available device–deployment plan pairs, such that the sum of local suitabilities $\boldsymbol{ls_{i,j}}$, i.e. the global suitability of the system $\boldsymbol{gs}$, is maximised:

$$maximise \sum_{i=1}^{m} \sum_{j=1}^{n} ls_{i,j} \qquad (1)$$

Since a deployment plan represents a whole set of software components to be deployed and executed on a device, it is assumed that any device can be assigned with at most one deployment plan at a time. At the same time, the same deployment plan can be assigned to multiple devices (provided they are well-suited for that). However, there are often scenarios, where the edge application provider might want to

limit the maximum number of devices assigned with the same deployment plan. For example, in DevOps, A/B testing is often implemented to test newly added features on a limited sub-set of staging devices before pushing them to production. Another example is the artificial *software diversity* – i.e., a common technique to improve system resilience, robustness and security by deploying functionally-equivalent, yet different software versions on multiple systems. In all such cases, updated software versions are required to be deployed only on a specific sub-set of the device fleet.

Assuming that all devices are equivalent in terms of their weights ($\forall i, j: w_{ij} = 1$), such a constraint is expressed through the deployment capacity c_j, which indicates the maximum number of devices that a deployment plan can be assigned to. As a result, the above maximisation integer program in Eq. 1 above should be solved subject to the following additional constraint:

$$\sum_{j=1}^{n} w_{ij} \leq c_j, \quad \text{where} \quad i = 1, \dots, m \qquad (2)$$

In addition to the main functional goal for distributing deployment plans across a fleet of devices, there is also non-functional, yet practical requirement of minimising the modelling complexity. This research has been conducted in collaboration with an edge software provider and reflects real business challenges faced by the company. In particular, it is important to propose a solution that would not require excessive knowledge and skills in logical programming to define assignment rules, so as to be naturally adopted by a domain expert, who is not necessarily a professional programmer. As we further discuss it in the next section, there exist approaches, which, albeit addressing the outlined requirements, are hardly usable in practice due to their modelling complexity and demanding requirements in terms of background knowledge and skills.

4.3.5.2 Fleet assignment through a running example

We again use a motivating example extracted from the TellU Use Case to explain the fleet assignment problem, and after that explain approach to address this problem. For simplicity and clarity, we focus on the three main hardware properties representing the device context (see *Table 9*) as deployment parameters affecting the assignment solving.

Fleet assignment at the edge falls into the category of combinatorial optimisation problems, in which the distribution of tasks to agents has to be solved following a global goal, such as maximisation of profit. This in turn requires that all possible assignment pairs have their own profit values, so that the maximum global profit can then be calculated. Accordingly, given the outlined requirements for an envisioned fleet assignment solution, as well as the limitations of the existing works, the proposed approach puts the assignment problem into the edge fleet context, and operates on a 5×5 matrix of sample devices and deployment plans, listed in *Table 10*.

Figure 45 depicts how this matrix is updated through the three main steps of the proposed approach, as described below.

Table 9. Deployment parameters.

Parameter	Description
CPU (MHz)	Minimum frequency of the installed CPU chip.
RAM (MB)	Minimum amount of memory available on-board.
Storage (GB)	Minimum amount of available disk space.

Table 10. Sample deployment plans and devices.

		CPU	RAM	Storage
Deployment	dp_1	1000	512	4

	w_1	0.3	0.3	0.4
	dp_2	1000	512	4
	w_2	0.1	0.1	0.8
	dp_3	900	1024	8
	w_3	0.3	0.4	0.3
	dp_4	700	1024	16
	w_4	0.2	0.2	0.6
	dp_5	1100	1024	8
	w_5	0.4	0.4	0.2
Devices	d_1	1000	512	16
	d_2	1200	1024	32
	d_3	1500	1024	32
	d_4	1000	512	8
	d_5	900	1024	16

	d_1	d_2	d_3	d_4	d_5
dv_1	1	1	0	0	0
dv_2	1	1	1	1	1
dv_3	1	1	1	1	1
dv_4	1	1	0	0	0
dv_5	0	0	1	1	0

(a) Filtering matching devices.

	d_1	d_2	d_3	d_4	d_5
dv_1	156	34	0	0	0
dv_2	268	65	380	132	546
dv_3	287	66	406	138	596
dv_4	119	20	0	0	0
dv_5	0	0	283	82	0

(b) Calculating local suitabilities.

	d_1	d_2	d_3	d_4	d_5
dv_1	156	34	0	0	0
dv_2	268	65	380	132	546
dv_3	287	66	406	138	596
dv_4	119	20	0	0	0
dv_5	0	0	283	82	0

(c) Calculating global suitabilities.

d_1	D_1
d_2	D_4
d_3	D_2
d_4	D_5
d_5	D_3

$gs = 1234$

(d) Assignment results.

Figure 45. Fleet assignment in three steps.

Step 1: Filtering Matching Devices

At the beginning of the assignment process, the system administrator is required to specify deployment parameters. It is expected that many of them are expressed as Boolean or enumerated data types and are therefore easy to check. As far as numeric parameters are concerned, it is required to specify a lower threshold – i.e., a minimum value for a corresponding device property that needs to be present in order for a device to qualify for the current deployment. It is assumed that if and only if all deployment parameters are matched by corresponding device properties, the device is selected for further assignment. This kind of filtering is a rather trivial task (and is essentially the current state of practice), which only requires evaluating the simple comparison (**GEQ**) and logical (**AND**) operators. As depicted in Figure 45, as an outcome of this operation, the device-deployment matrix is populated with Boolean values indicating whether or not a device matches a deployment.

Step 2: Calculating Local Suitabilities

Albeit a Boolean **TRUE** already suggests that a specific deployment plan can be installed on a given device, it does not provide a means of doing so in an optimum way, since multiple device-deployment plan alternatives have essentially the same `weight'. Therefore, it is required to quantify a *local suitability* of a device for a deployment plan, given the numerically expressed context properties.

To do so, we apply the **Weighted Product Method (WPM)**, which is a well-known technique in the domain of solving multi-criteria decision problems [32]. The WPM is acknowledged to be a simple and light-weight, yet efficient approach [33], which allows comparing alternatives in terms of multiple criteria, not necessarily expressed using the same units (as opposed to a similar, yet less applicable Weighted Sum Method [34]). In the context of edge fleet assignment, this means that deployment parameters can include rather heterogeneous context properties, such as hardware characteristics, network signal levels, battery charge levels, or GPS coordinates.

Accordingly, given multiple deployment parameters, calculating a local suitability of a device for a deployment can be seen as a multi-criteria decision problem. In our context, the problem is defined on $\boldsymbol{m}$ alternatives (i.e., devices), from $\boldsymbol{A_1}$ through $\boldsymbol{A_m}$, and $\boldsymbol{n}$ decision criteria (i.e., deployment parameters matched by device properties), from $\boldsymbol{C_1}$ through $\boldsymbol{C_n}$. The WPM assumes that each criterion $\boldsymbol{C_j}$ has a relative weight of importance $\boldsymbol{w_1}$ (also present in Table *10*), whereas the property of device A_i when it is evaluated in terms of parameter $\boldsymbol{C_j}$ is denoted as $\boldsymbol{a_{ij}}$. As a result, the local suitability of a device for a deployment plan is calculated as follows:

$$P(A_i) = \prod_{j=1}^{n} \left(a_{ij}\right)^{w_j}, \quad \text{for} \quad i = 1, \dots, m \quad \text{subject to} \quad \sum_{j=1}^{n} w_j = 1 \qquad (3)$$

This operation is repeated for each of the $\boldsymbol{m}$ devices, resulting in an array of local suitability values calculated for a single deployment plan. Next, the same process is repeated for the rest of the plans, while the previously filled **TRUE** values in the matrix in Figure 45(b) are replaced with integers, representing a local suitability for each device-deployment plan pair. It is worth noting that the classical WPM approach assumes that the local suitability is a maximisation function, where the higher values the better. In practice, there might be scenarios, where, on contrary, minimum values are beneficial. For example, it might be required to deploy staging software updates to devices near the vendor's office (i.e., the shorter distance the better) for faster manual access. Such cases can be implemented by inverting the desired parameters.

Step 3: Calculating Global Suitability

Next, given local suitability values, among all device-deployment plan pairs we find the final assignment pairs, aiming for a maximum *global suitability*, such that deployment plans are assigned to the most suitable devices. We calculate the global suitability, treating it as an assignment problem (see Figure 45(c)). Given the constraint that a device can only be assigned with a single plan at a time, as an outcome of this operation, the assignment matrix will be finally reduced to a device-deployment plan map, ready for final enactment. Figure 45(d) depicts the final pairs and the suggested assignment's global suitability $\boldsymbol{gs}$=1234.

The assignment problem falls into the category of linear integer problems. They are well studied and the research community has come up with many heuristics-based and approximation approaches for solving such problems [35]. Furthermore, there also exists several solvers [36] – i.e., highly-optimized and reliable software implementations in many programming languages for efficiently solving such linear integer problems.

4.3.5.3 Benefits of the approach for DevOps engineers

Among the main benefits of this approach, which are specifically relevant to the DevOps practices and thus would contribute to the continuous development and release of software to edge devices, are the following:

- **Low modelling complexity**: it was important to try a solution that would not require yet another programming language or logical formalism to model the fleet and define the assignment logic. Some parameterisation and tuning were acceptable, provided it would be done via a graphical interface, rather than through the code. As a result, the developed approach only requires annotating deployment plan parameters with weights, which is implemented as a sliding visual component in the user interface. Another related benefit in this respect is the WPM's intrinsic support for normalisation of cross-dimensional criteria. That is, within a single assignment scenario, it is possible to compare parameters with different scales and units of measurement.
- **Scalability**: as demonstrated by the experiments (included in the related paper), the implemented proof of concept is able to calculate assignments across hundreds of devices and deployments within seconds – a reasonable time frame, which enables responsive online interaction with the assignment engine and allows trying out multiple assignment alternatives.
- **Extensibility**: even though the presented research primarily focused on an e-Health scenario, it is not hard-coded to this specific domain and the set of parameters and can be seamlessly extended to accommodate different application domains. The list of parameters for a deployment on the client interface is populated dynamically from a JSON file, which, if needed, can be extended with other fields to quickly react to emerging business requirements (e.g., in the future, it might be required to take into consideration available cooling facilities before assigning computationally-intensive software). The extensibility is especially important for a heterogeneous fleet, where newer devices with previously unseen characteristics are continuously added to the system.

4.3.6 Tutorial

The source code of DivEnact is hosted in the GitLab repository: https://gitlab.com/enact/divenact

The tutorial for using the tool is provided in the README.md file in the root folder of this repository. The tutorial provides a step-by-step instruction of launching and using the full-functioned DivEnact service, including the provisioning of a backend Azure IoT Hub service and the bootstrapping of the IoT/Edge devices. In addition, we also provide a demo version without Azure IoT Hub service nor any physical devices, so that users can have a quick trial of the core technical part of DivEnact (i.e., the fleet assignment between software and devices).

4.3.7 Main innovation

Based on our systematic literature review [37], DivEnact is the first approach that extends the concept of automatic software deployments from specific resources to a fleet of resource groups, to enable DevOps on the production-phase IoT applications which comprise a large number of similar device groups. Automatic software deployment, or Infrastructure as Code (IaC) is the corner stone of DevOps, which liberate developers from manual software deployment by automatically executing the pre-defined deployment models or scripts. Existing solutions, such as Ansible, Chef, and GeneSIS, all require developers specify the exact resources in the deployment model. DivEnact goes one step further and allows developers to specify deployment model on an abstract resource group, and then assign a set of these models automatically to a fleet of concrete resource groups.

To solve the problem of assigning multiple deployment plans to many resources within a fleet, we experimented with a model-driven engineering (MDE) approach and utilized research tools such as constraint solving and resource assignment. The result is promising and is well-acknowledged by the research community.

In summary, the research around DivEnact has led to the following articles published in scientific venues, and we have also other papers in the pipeline.

- Hui Song, Rustem Dautov, Nicolas Ferry, Arnor Solberg and Franck Fleurey. 2020. Model-based fleet deployment of edge computing applications. Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. Association for Computing Machinery, 132–142. DOI:https://doi.org/10.1145/3365438.3410951 (Best paper award)
- Dautov, R., Song, H. and Ferry, N., A Light-Weight Approach to Software Assignment at the Edge., in the 9th International workshop on cloud and edge computing and application management, in conjunction with 13th IEEE/ACM utility and cloud computing conference (UCC). 7-10 December 2020 - Leicester, UK.
- Dautov, R. and Song, H., 2019. Towards IoT Diversity via Automated Fleet Management. In MDE4IoT/ModComp@ MoDELS (pp. 47-54).

5 Conclusions

The goal of WP4 in ENACT is to provide support to define and ensure the secure, resilient and privacy-aware behaviour of Smart IoT Systems (SIS). The work package deals with security and privacy support to development and operations phases of the DevOps cycle. At development phase the solutions offered mainly include system security, privacy and resilience (diversity) requirements specification mechanisms together with the associated controls specification. At operations phase, the support is focused on continuous monitoring of possible security and privacy incidents and attacks to the IoT system as well as early reaction to them.

The main objective of this report is the description of the design and implementation of the following enablers which are the core of the work in WP4: i) a Security and Privacy Monitoring and Control enabler, which includes an advanced context-aware access control (CAAC) mechanism and ii) a Diversifier to be able to analyse diversity requirements of SIS and request software variants deployments when needed.

The document describes the details of the delivered software prototypes of the Enablers above and presents the operation of the different methods and tools and how they can be used in the ENACT framework. The report offers the analysis of which SIS properties related to trustworthiness are supported by WP4 tools and how they are addressed by each of the tools. Furthermore, the final coverage of Use Cases' requirements is provided. As described in the Table 7, all the initial requirements related to security, privacy and resilience identified by the Use Cases have been finally addressed.

Large-scalability aspect of IoT systems is a characteristic considered from the design phase of the WP4 Enablers, and final prototype tools fully support scalability. In particular, DivEnact provides a direct answer to scalability concern of GeneSIS as the ENACT deployment solution: instead of defining a huge GeneSIS model covering all the devices in a large fleet, it is recommended to divide the fleet into subsystems with similar compositions. In this way, one only need to define a number of small GeneSIS models for the different types of subsystem and use DivEnact to manage the assignment of GeneSIS models to the right subsystems. Furthermore, the tools are seamlessly integrated among them and with other ENACT enablers such as GeneSIS and Root Cause analysis tool. For further details of final integration please refer to deliverable *D5.4 ENACT DevOps Framework- Final version*.

The usefulness and efficiency of the Enablers has been evaluated in a series of tests within the ENACT Use Cases and the results of such evaluation are provided in *D1.5 Final evaluation and validation report* of ENACT.

Several research lines related to SIS security and privacy support in DevOps remain open for further improvement after ENACT project, as follows.

On continuous monitoring and detection of security and privacy anomalies, Tecnalia keeps investigating AI-based techniques to improve the detection of advanced or zero-day cyberattacks and plans to further research on automated intelligent ways for security event classification. Smart root cause analysis and automated security orchestration are also areas which offer still unexplored research room and are necessary to tackle towards full compliance assurance and certification under future cybersecurity certification schemes of EU Cybersecurity Act.

On Context-aware Access Control, after the ENACT project, EVIDIAN will integrate the CAAC as a new feature in its IDaaS solution to extend the access control capabilities to connected devices. Special attention will be paid to the scalability of the solution. The simple risk server used to demonstrate the context-awareness capabilities of the CAAC will be extended and improved to become a component of a new future module "Evidian Prescriptive IAM" whose infrastructure will have been validated in the context of the ENACT project. In this objective, further investigations on risk models in the identity and access management context will be carried out, in order to develop a prescriptive enforcement of authentication and authorization decisions, based on risk indicators.

On diversity applied to fleet deployment and operation, SINTEF and TellU will keep investigating and developing the concept of fleet deployment after the ENACT project, under a new innovation project named FLEET, funded by the Research Council of Norway. The primary objective of the subsequent project is to implement the concept into the DevOps practice of TellU's production line. Along with the TellU Use Case as an IoT/Edge application, we also introduce a second use case in the FLEET project, i.e., the Dolittle microservice operation platform, to extend the scope of the fleet deployment concept. From the research point of view, the new project will extend the work in ENACT in the following directions:

- Automatic and headless provisioning of IoT/Edge devices and microservice environments.
- Fleet-level monitoring and anomaly detection.
- Improving fleet assignment and monitoring policies using machine learning.

6 References

Bibliography

[1] Noguero, A., Rego, A., & Schuster, S. Towards a Smart Applications Development Framework. Social Media and Publicity, 27.

[2] Rios, E., Iturbe, E., & Palacios, M. C. (2017, August). 'Self-healing Multi-Cloud Application Modelling'. Proc. Int. Conf. on Availability, Reliability and Security (p. 93). ACM.

[3] MUSA EU H2020 project, Deliverable D2.3, https://www.musa-project.eu/sites/musa3.drupal.pulsartecnalia.com/files/documents/MUSA%20D2.3%20Final%20SbD%20methods%20for%20multi-cloud%20applications.pdf.

[4] National Institute of Standards and Technology (NIST), 'Security and Privacy Controls for Information Systems and Organizations'. NIST SP-800-53, revision 4.

[5] ISO/IEC 27002:2013 — Information technology — Security techniques — Code of practice for information security controls (second edition)

[6] ISO/IEC 27017:2015 Information technology -- Security techniques -- Code of practice for information security controls based on ISO/IEC 27002 for cloud services

[7] ISO/IEC 27018:2014 — Information technology — Security techniques — Code of practice for protection of Personally Identifiable Information (PII) in public clouds acting as PII processors

[8] Cloud Control Matrix (CCM) Alliance, C.S.: Cloud security alliance, cloud controls matrix v3.0.1 (11-12-18 Update), https://cloudsecurityalliance.org/group/cloud-controls-matrix/

[9] Cloud Security Alliance: 'Consensus Assessments Initiative Questionnaire v3.0.1 (9-1-17 Update)', https://cloudsecurityalliance.org/download/consensus-assessments-initiative-questionnaire-v3-0-1/

[10] The OWASP Application Security Verification Standard (ASVS) Project, https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

[11] Database Hardening Best Practices, Berkeley Information Security Office, https://security.berkeley.edu/resources/best-practices-how-articles/database-hardening-best-practices

[12] National Institute of Standards and Technology (NIST), 'Security and Privacy Controls for Information Systems and Organizations'. NIST SP-800-53, revision 5 Draft.

[13] Morin, Brice, Jakob Høgenes, Hui Song, Nicolas Harrand, Benoit Baudry. Engineering Software Diversity: a Model-Based Approach to Systematically Diversify Communications. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 155-165. ACM, 2018.

[14] ISO/IEC PRF 27552 Security techniques -- Extension to ISO/IEC 27001 and ISO/IEC 27002 for privacy information management -- Requirements and guidelines [Under development]

[15] Mockel, C. & Abdallah, A. (2010). "Threat modeling approaches and tools for securing architectural designs of an e-banking application". Sixth International Conference on Information Assurance and Security (IAS), 149-154, doi: 10.1109/ISIAS.2010.5604049.

[16] Department of Homeland Security. Cyber Threat Modelling: Survey, Assessment, and Representative Framework. April, 2018. Authors: Deborah J. Bodeau, Catherine D. McCollum, David B. Fox.

[17] Jing, Q.; Vasilakos, A.V.; Wan, J.; Lu, J.; Qiu, D. Security of the Internet of Things: Perspectives and challenges. *Wirel. Netw.* 2014, *20*, 2481–2501.

[18] Wu, M.; Lu, Ti.; Ling, Fe.; Sun, J.; Du, H. Research on the architecture of Internet of things. In Proceedings of the 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Chengdu, China, 20–22 August 2010; Volume 5, p. 484-487

[19] E. Rios, A. Rego, E. Iturbe, M. Higuero, and X. Larrucea, Continuous Quantitative Risk Management in Smart Grids Using Attack Defense Trees. MDPI Sensors, vol. 20, no. 16, p. 4404, Aug. 2020

[20] Avizienis, Algirdas. The N-version approach to fault-tolerant software. In IEEE Transactions on software engineering 12 (1985): 1491-1501.

[21] Brice Morin, Jakob Høgenes, Hui Song, Nicolas Harrand, and Benoit Baudry. 2018. Engineering Software Diversity: a Model-Based Approach to Systematically Diversify Communications. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS '18). ACM, New York, NY, USA, 155-165. DOI: https://doi.org/10.1145/3239372.3239393

[22] IETF, The Wire Image of a Network Protocol. https://tools.ietf.org/html/rfc8546

[23] Industrial Internet Consortium. Industrial Internet of Things Volume G4: Security Framework. IIC:PUB:G4:V1.0:PB:20160919.

[24] The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749

[25] OpenID Connect. https://openid.net/connect/

[26] Anind K. Dey Understanding and Using Context http://www.kevinli.net/courses/mobilehci_w2014/papers/dey-context-01.pdf

[27] Grammatikis, P.R., Sarigiannidis, P., Iturbe, E., Rios, E., Sarigiannidis, A., Nikolis, O., Ioannidis, D., Machamint, V., Tzifas, M., Giannakoulias, A. and Angelopoulos, M., 2020, June. Secure and Private Smart Grid: The SPEAR Architecture. In 2020 6th IEEE Conference on Network Softwarization (NetSoft) (pp. 450-456). IEEE.

[28] Hui Song, Rustem Dautov, Nicolas Ferry, Arnor Solberg, and Franck Fleurey. 2020. Model-based fleet deployment of edge computing applications. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS '20). ACM, 132–142. DOI:https://doi.org/10.1145/3365438.3410951.

[29] Zhang, L. and Malik, S., 2002, July. The quest for efficient boolean satisfiability solvers. In International Conference on Computer Aided Verification (pp. 17-36). Springer, Berlin, Heidelberg.

[30] Clark Barrett and Cesare Tinelli. 2018. Satisfiability modulo theories. In Handbook of Model Checking. Springer, 305–343.

[31] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 337–340.

[32] Bridgman, Percy Williams. Dimensional analysis. Yale university press, 1922.

[33] Aminudin, Nur, et al. "Weighted product and its application to measure employee performance." International Journal of Engineering & Technology 7.2.26 (2018): 102-108; Mateo, José Ramón San Cristóbal. "Weighted sum method and weighted product method." Multi criteria analysis in the renewable energy industry. Springer, London, 2012. 19-22.

[34] Tofallis, Chris. "Add or multiply? A tutorial on ranking and choosing with multiple criteria." INFORMS Transactions on education 14.3 (2014): 109-119.

[35] Osman, Ibrahim H. "Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches." Operations-Research-Spektrum 17.4 (1995): 211-225; Linderoth, Jeff T., and Martin WP Savelsbergh. "A computational study of search strategies for mixed integer programming." INFORMS Journal on Computing 11.2 (1999): 173-187; Geoffrion, Arthur M., and Roy EARL Marsten. "Integer programming algorithms: A framework and state-of-the-art survey." Management Science 18.9 (1972): 465-491.

[36] Genova, Krasimira, and Vassil Guliashki. "Linear integer programming methods and approaches–a survey." Journal of Cybernetics and Information Technologies 11.1 (2011); Fourer, Robert. "Linear programming." OR/MS Today 40.3 (2013): 40-53.

[37] Nguyen, P., Ferry, N., Erdogan, G., Song, H., Lavirotte, S., Tigli, J.Y. and Solberg, A., 2019, July. Advances in deployment and orchestration approaches for IoT-a systematic review. In 2019 IEEE International Congress on Internet of Things (ICIOT) (pp. 53-60). IEEE.

Software tool repositories

[38] Open Source IDS / IPS / NSM engine Suricata, rules format documentation. https://suricata.readthedocs.io/en/suricata-4.1.4/rules/intro.html

[39] Elastic stack products, https://www.elastic.co/products/

[40] T-shark, Core network protocol analyzer of Wireshark, https://www.wireshark.org/docs/man-pages/tshark.html

[41] Elastic Filebeat Software, https://www.elastic.co/en/products/beats/filebeat

[42] Elastic Auditbeat, https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html

[43] Open Source IDS / IPS / NSM engine Suricata, https://suricata-ids.org/

[44] Open Source IDS / IPS / NSM engine Suricata ruleset update management documentation, https://suricata.readthedocs.io/en/suricata-4.1.4/rule-management/suricata-update.html

[45] Apache kafka, https://kafka.apache.org/

[46] Elastic Logstash Software, https://www.elastic.co/en/products/logstash

[47] Elasticsearch Software, https://www.elastic.co/en/products/elasticsearch

[48] Elastic Kibana Software, https://www.elastic.co/en/products/kibana

[49] Apache Kafka clients, https://cwiki.apache.org/confluence/display/KAFKA/Clients

[50] About Docker CE installation documentation, https://docs.docker.com/install/

[51] MongoDB, https://www.mongodb.com/

[52] CouchDB, https://couchdb.apache.org/

[53] Ant.Design, https://ant.design/

[54] Node.js, https://nodejs.org/en

[55] Ngrok, https://ngrok.com/

[56] Google Coral, https://coral.ai/products/accelerator

[57] Apache NiFi, https://nifi.apache.org/

[58] Quarkus, https://quarkus.io/